FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Real-Time Filtering of Monte Carlo Noise on GPU

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering & Internet Computing

by

Maxim Davletaliyev
Registration Number 1229133

to the Faculty of Informatics

at the TU Wien

Advisor: Assoc. Prof. Dr. Hannes Kaufmann
Assistance: Mag. Dr. Peter Kán

Vienna, 22nd August, 2016

Maxim Davletaliyev	Hannes Kaufmann

Erklärung zur Verfassung der Arbeit

Maxim Davletaliyev
Lorenz-Müller-Gasse 1a/5219

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 22. August 2016

Maxim Davletaliyev

Acknowledgements

First and foremost, I would like to express my gratitude to Peter Kán for his insightful ideas, continuous guidance, support and patience. This thesis would not have been completed or written without him. A special thanks is addressed to Hannes Kaufmann for his professional evaluation.
Furthermore, I would like to thank the whole VR Group team for showing me the beauty and fun of the computer graphics and virtual reality. I am really grateful to Peter Kán, Khrystyna Vasylevska and Iana Podkosova for providing comfortable and friendly working atmosphere during this master thesis.
I would like to acknowledge the authors of the 3D models: Oliver Deussen, Guillermo M. Leal Llaguno, Marko Dabrovic and Mihovil Odak for their courtesy of allowing the utilization of their scenes.
Finally, I must express my very profound gratitude to my parents for the unconditional support throughout my studies.

Kurzfassung

Die Herstellung von fotorealistischen Bildern, die kaum von realen Bildern unterscheidbar sind, ist eines der wichtigsten Probleme in der Computergraphik. Übermäßige Abtastung während der Monte Carlo Integration bei Physically Based Rendering und besonders bei der Monte Carlo Path Tracing ermöglichen Bilder dieser Qualität. Das Problem der Monte Carlo Integration ist die Varianz bei einer niedrigen Abtastrate, welche als Rauschen im Endbild erscheint. Um dieses Problem zu umgehen, wird mehrdimensionale Filterung verwendet.

In dieser Arbeit wird die Anwendbarkeit der Genetischen Programmierung für die Suche nach neuen mehrdimensionalen Filterausdrücken untersucht. Außerdem werden drei neue Ausdrücke vorgestellt, die bei unserer Methode generiert wurden. Unsere Methode besteht aus iterativen zufälligen Änderungen der ursprünglichen Ausdrücke bis zur Erfüllung des Abbruchkriteriums und aus dem Vergleich der mit neu erzeugten Ausdrücken erhaltenen gefilterten Pixelwerte mit den Ground Truths der Trainingsszenarios. Die so erhaltenen Ausdrücke erzielen bessere Ergebnisse als ein Crossbilateraler Filter mit konstanten Parametern. Desweiteren erlaubt unsere GPU Implementierung der identifizierten Ausdrücke eine schnelle Filterung des Monte Carlo Rauschens mit einer Rechenzeit von weniger als einer Sekunde.

Abstract

Producing photo-realistic images, hardly distinguishable from the real photos, is one of the most important problems in computer graphics. Physically based rendering and particularly Monte Carlo path tracing is able to produce images of such quality by performing excessive sampling during Monte Carlo integration. The problem of Monte Carlo Integration is a high variance at low sampling rate. This variance appears as a noise in final image. In order to address such problem high-dimensional filtering is used.

In this thesis we inspect the applicability of the Genetic Programming for the search of new high-dimensional filtering expressions and present three novel expressions generated by our method. Our method consists of iterative random changes of initial expressions until the finishing criterion is met and the comparison of the filtered pixel values, obtained with newly generated expressions, with the ground truth of the training scenes. The resulting expressions perform better than cross-bilateral filter with constant parameters. Additionally, our GPU implementation of identified expressions allows fast filtering of Monte Carlo noise with computational time of less than a second.

Contents

Kurzfassung vii

Abstract ix

Contents xi

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Problem Statement . . . 2
- 1.3 Contributions . . . 2
- 1.4 Thesis Outline . . . 4

2 Theoretical Background and Related Work 5
- 2.1 Theory of Light Transport . . . 5
  - 2.1.1 Rendering Equation . . . 5
  - 2.1.2 Bidirectional Reflectance Distribution Function . . . 8
- 2.2 Physically Based Algorithms for Global Illumination . . . 9
  - 2.2.1 Ray Tracing . . . 9
  - 2.2.2 Monte Carlo Rendering . . . 10
    - 2.2.2.1 Monte Carlo Integration . . . 10
    - 2.2.2.2 Path Tracing . . . 11
    - 2.2.2.3 Bidirectional Path Tracing . . . 12
    - 2.2.2.4 Russian Roulette . . . 12
- 2.3 Filtering . . . 13
  - 2.3.1 General Image Denoising Algorithms . . . 14
  - 2.3.2 Filtering of Monte Carlo Noise with Adaptive Sampling . . . 15
  - 2.3.3 Filtering of Monte Carlo Noise without Adaptive Sampling . . . 16
- 2.4 Genetic Programming . . . 16
  - 2.4.1 Program Representation . . . 17
  - 2.4.2 Population Initialization . . . 17
  - 2.4.3 Termination Criterion . . . 18
  - 2.4.4 Selection . . . 18
  - 2.4.5 Control Parameters Selection . . . 19

2.4.6 Expansion . . . 20
2.5 Genetic Programming in Computer Graphics . . . 20

3 Generation of New Filtering Expressions with Genetic Programming 21
3.1 Overview of the Search Algorithm . . . 21
3.2 Genetic Programming for Filtering . . . 22
3.2.1 Operators . . . 22
3.2.1.1 Functional Operators . . . 22
Unary Operators . . . 22
Binary Operators . . . 22
Vector Operators . . . 23
3.2.1.2 Terminal Variables . . . 24
Vector Variables . . . 24
Scalar Variables . . . 25
Constants . . . 25
3.2.2 Codebook . . . 26
3.2.3 Fitness Function . . . 28
3.2.4 Expansion Operators . . . 29
3.2.5 Initial Population . . . 30
3.2.6 Parameters Selection . . . 31
3.2.7 Input . . . 32
3.3 Implementation . . . 32
3.3.1 Hardware . . . 32
3.3.2 Input Generation . . . 32
3.3.3 Representation of Abstract Syntax Tree . . . 34
3.3.4 Evaluation of Expressions . . . 35

4 Results 37
4.1 Performance Evaluation . . . 37
4.1.1 CPU vs GPU . . . 37
4.1.2 Perfromance Measurements . . . 38
4.1.3 Performance Discussion . . . 40
4.2 Error Evaluation . . . 40
4.3 Empirical Investigation of Features . . . 42
4.4 Qualitative Comparison of the Results . . . 43
4.5 Discussion . . . 45

5 Limitations and Future Work 49

6 Conclusion 53

Bibliography 55

CHAPTER 1

Introduction

1.1 Motivation

The main objective of computer graphics is generation of high-quality images from the 3D models. This process is actively used in such fields as video game industry, movie industry and medical visualization. The video game industry achieved incredible heights nowadays. The amount of turnover in this industry is the source of big investments in software and hardware, especially in graphic cards. Another field that is being excessively evolving is virtual reality. With the help of the headset the full immersion in the 3D world is possible. All of these fields require high degree of interactivity, what leads to the demand for fast reaction time to the user's input. Rasterization is a rendering technique that is able to provide required degree of interactivity. Nowadays new games want to allure customers by the incredible graphics and level of details. The same demand holds for the VR, the headsets are getting higher resolution and starting to require photo-realistic images. Unfortunately, rasterization techniques are only able to render scenes with simplified illumination.

In contrast to that, physically based rendering is able to produce images that can be hardly distinguishable from the real photos. It uses the approximation of the Maxwell's equations called rendering equation (see Section 2.1.1). Rendering equation accounts for the light coming from all possible directions and therefore supports global illumination. The problem is that rendering equation is the Fredholm integral equation of the second kind and therefore infinite dimensional integral. Such integral can not be solved analytically except for very simple cases. In order to address such problem the numerical solutions are used, good example of which is Monte Carlo (MC) integration. MC integration uses samples (values of the functions at some randomly chosen directions or points) to approximate the integral. To obtain a visually pleasant photo-realistic image a lot of samples are needed. For the scene of medium complexity around 30 thousands samples per pixel are required to produce noise-free images. Figure 1.1 shows the impact

of different sampling rates. The rendering of an image with high amount of samples takes minutes or even hours. With the aforementioned rendering time pure MC rendering can not be used in the real-time applications. There are several methods for mitigation of such problem: importance sampling, adaptive sampling and filtering (Section 2.3) of the noise. This thesis is concentrated on the filtering of Monte Carlo noise in the Path Tracing.

1.2 Problem Statement

With the low number of samples Monte Carlo method introduces noise in the image (see Figure 1.1a). The source of this noise is the variance from the Monte Carlo estimator, because with small amount of samples estimator underestimates or overestimates the integral in the Rendering Equation. Such noise is unpleasant for the eye, therefore, Monte Carlo method at low sampling rate is not utilizable.

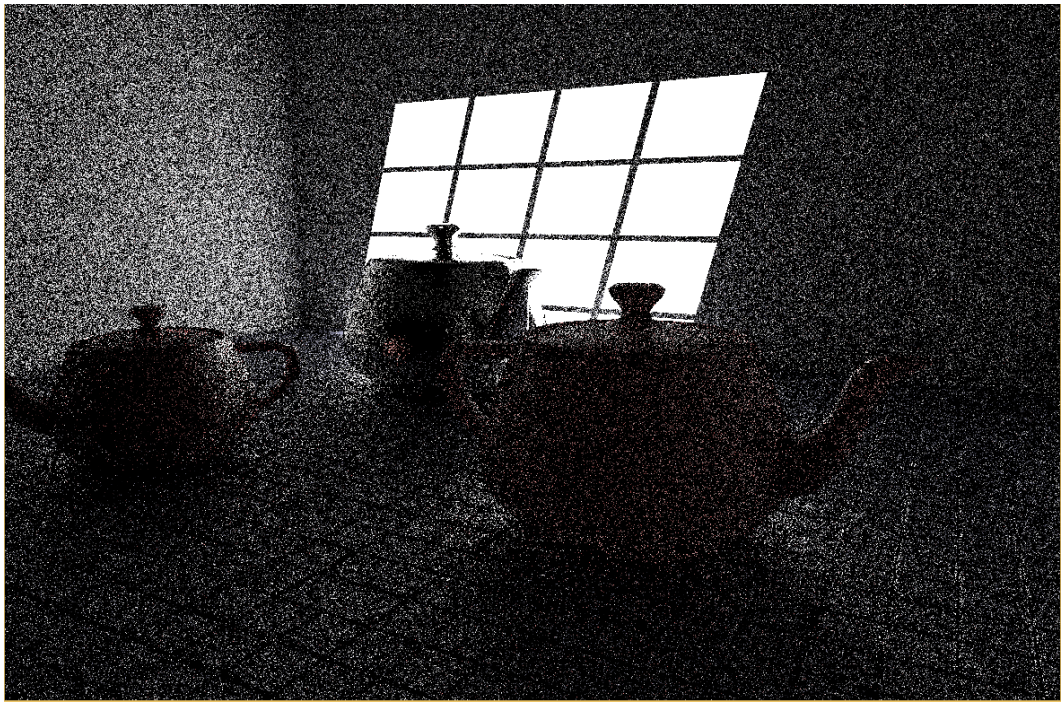
The main goal of this thesis is to find new approaches for Monte Carlo noise reduction with the help of image-space high-dimensional filtering. We search for new filtering expressions by utilizing Genetic Programming (GP). This thesis searches for the answers to the following research questions:

1. Is Genetic Programming capable of inferring new filtering expressions, which can outperform current state of the art?
2. Are there any good parameters or sets of data that can give better final results?
3. What is the best set of auxiliary information (for example normal, world position, etc.) for the filters?

1.3 Contributions

This thesis addresses the problem of filtering the Monte Carlo noise in the physically based rendering by the novel approach: by generating new expressions with the assistance of GP algorithm. The main contributions of this thesis are:

- novel filtering expressions. A set of generated expressions that filter noise from the image rendered at low sampling rate.
- investigation of the GP algorithm's applicability for the Monte Carlo filtering with the framework for searching of new expressions.
- empirical investigation of the importance of scene features.
- a proof of concept implementation for the search of new expressions and for the GPU filtering with such expressions.



(a) Scene rendered with 4 samples per pixel. The rendering time of the scene is 4 seconds.

(b) Scene rendered with 96 000 samples per pixel. The rendering time of the scene is 6 hours.

Figure 1.1: Figure illustrates the differences between undersampled scene (a) and a scene rendered at the sampling rate required for the noise free image (b). The images were rendered with the PBRT [PH10].

1.4 Thesis Outline

The thesis continues as follows. Theoretical background and recent research on this topic are discussed in Chapter 2. The core concept of the algorithm and implementation specifications are described in Chapter 3. The evaluation and results are shown in Chapter 4. The limitations of the algorithm and possible future improvements are presented in Chapter 5. Chapter 6 concludes the thesis.

CHAPTER 2

Theoretical Background and Related Work

2.1 Theory of Light Transport

Light is essential for human vision. Only trough the light we can perceive the world around us. Light falls at the pupil of our eye, brain processes neural signals originated from the eye and gives us the picture of the world. Therefore, to be able to produce photorealistic image, the simulation of the light is required and a process of tracing the light rays is a corner stone for rendering in computer graphics. Light should be traced throughout all the way from start, the light source, then as it is reflected, refracted from the objects' surfaces and until it ends up on the camera's lens. With the rendering equation it is possible to perform such tracing of the light rays. Rendering equation is described in Section 2.1.1. Several algorithms (Section 2.2) are based on the evaluation of Rendering equation. This section describes mathematical model of light transport used in rendering.

2.1.1 Rendering Equation

Since the light is an electromagnetic wave it complies to the Maxwell's equations. Therefore, it is possible to model light transport by solving the Maxwell's equations. But to solve Maxwell's equations computations in terms of atoms should be performed, that are still infeasible for modern hardware. Instead of that, a macro-level optical model is preferable. Such model can be derived from the energy balance. Taking into account the energy conservation law in isolated systems it can be stated that the difference of incoming and outgoing energy should be equal to the difference between absorbed and emitted energy. Thus, we are interested in the amount of energy that comes at specified angle and the amount of energy that exits the surface. The most often used radiometric

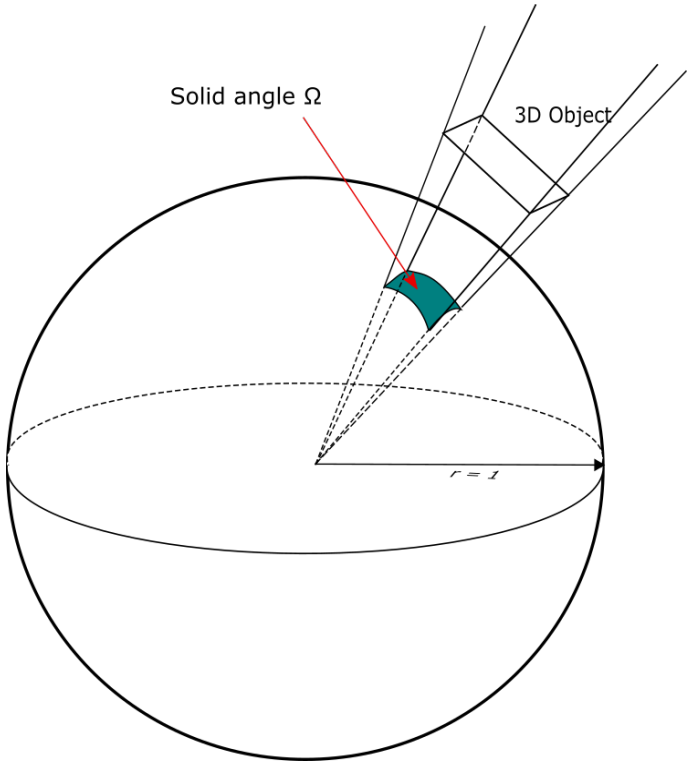


Figure 2.1: The solid angle Ω subtended by a 3D object, is equal to the surface area of its projection onto the unit sphere.

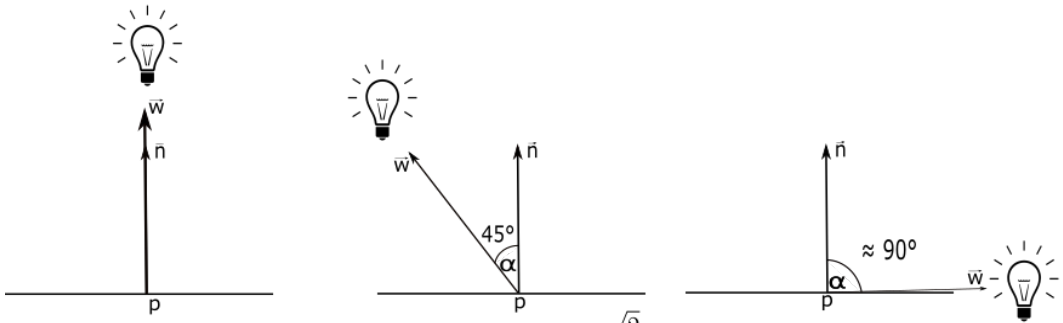
quantity to calculate these energies is radiance. Radiance is the total radiant flux received by the surface per unit solid angle per unit projected area. The solid angle (Figure 2.1) Ω subtended by a surface S is defined as the surface area Ω of a unit sphere covered by the surface's projection onto the sphere. It can be interpreted as a measure of how large the object looks to an observer. The radiant flux is the radiant energy received per unit time. The radiance coming from point p in direction $\vec{w}$ is denoted by $L(p, \vec{w})$. In order to ensure the energy balance, the radiance coming from the surface should be equal to the sum of emitted and reflected radiance

$$L_o(p, \vec{w}) = L_e(p, \vec{w}) + L_r(p, \vec{w})$$

All the light energy, that comes to the specific point on the surface, contributes to the radiance reflected from that point to the outgoing direction. To take this light energy into account we need to integrate over hemisphere for all possible incoming directions, what can be written as:

$$L_o(p, \vec{w}) = L_e(p, \vec{w}) + \int_{\mathcal{H}^2} f_r(p, \vec{w}, \vec{w}') L_i(p, \vec{w'}) \left|\vec{w}' \cdot \vec{n}\right| d\vec{w}' \tag{2.1}$$

where:



$|\vec{w} \cdot \vec{n}| = cos\alpha_{\{\alpha=0\}} = 1$ $|\vec{w} \cdot \vec{n}| = cos\alpha_{\{\alpha=45\}} = \frac{\sqrt{2}}{2}$ $|\vec{w} \cdot \vec{n}| = cos\alpha_{\{\alpha\approx 90\}} \approx 0$

Figure 2.2: The values of light attenuation at different angles to the light source.

- $L_o(p, \vec{w})$ is the outgoing radiance at position p in direction $\vec{w}$,
- $L_e(p, \vec{w})$ is the emitted radiance,
- $\mathcal{H}^2$ is the unit hemisphere subtended at position p in the direction of the surface normal $\vec{n}$,
- $f_r(p, \vec{w}, \vec{w'})$ is bidirectional reflectance distribution function (BRDF) described in Section 2.1.2,
- $L_i(p, \vec{w'})$ is the incident radiance from incoming direction $-\vec{w'}$,
- $|\vec{w'} \cdot \vec{n}|$ is the dot product between a light vector $\vec{w'}$ and surface normal. This product is denoted as a light attenuation. Attenuation can be intuitively understood as the loss of intensity with the increase of the angle between vector and surface normal. This visualization can be seen in Figure 2.2.

The rendering equation (Equation 2.1) was introduced by Kajiya [Kaj86]. With utilization of rendering equation it is possible to simulate global illumination and other distributed effects such as: motion blur, area light sources, soft shadows and depth of field. Since it is an optical model, that approximates Maxwell's equations, some simplifications have been made. The main simplification of this model is that it does not account for the fact that light is a wave. Therefore, Rendering Equation does not account for interference, diffraction and polarization. Another simplification is an assumption that the media between the objects are considered to be homogeneous and such effects as participating media with the fog as an example can not be easily simulated.

Rendering equation is a Fredholm integral equation of the second kind. Since the result is dependent on the scattered light, the integral has infinite dimensionality. The complexity also depends on the scene's geometry, different BRDF representations and different effects. It is impossible to solve Rendering equation analytically in general. Therefore, a numerical solution should be used instead.

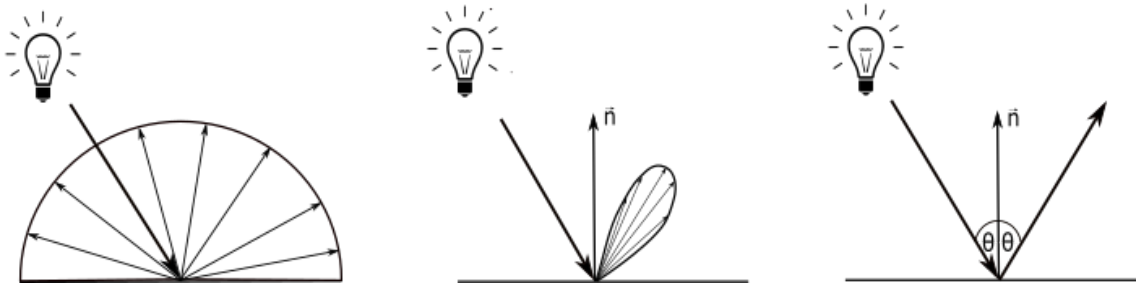


Figure 2.3: Left: Perfect diffuse reflection. Middle: Glossy reflection. Right: Perfect spectular reflection.

2.1.2 Bidirectional Reflectance Distribution Function

BRDF is defined as the ratio of the quantity of reflected light in direction $\vec{w}$ at the point p on the surface, to the amount of light that reaches the surface from direction $\vec{w'}$. It can be formulated mathematically in following way:

$$f_r(p, \vec{w}, \vec{w'}) = \frac{dL_r(p, \vec{w})}{dE(x, \vec{w'})}) = \frac{dL_r(p, \vec{w})}{L_i(p, \vec{w'})\,|\vec{w'} \cdot \vec{n}|\,d\vec{w'}}$$

Reflection from surfaces can be split into three broad categories:

- **Perfect diffuse** (Figure 2.3 left) - reflects light uniformly to all directions on the hemisphere.
- **Perfect specular** (Figure 2.3 right) - reflects light according to the law of reflection.
- **Glossy** (Figure 2.3 middle) - is something in between, it reflects the light mainly in one direction, but slightly spread around that direction. Perfectly diffuse and specular do not occur in real life, so most objects are glossy.

In order to be physically plausible the BRDF is required to conform to the following laws:

1. **Positivity**. BRDF is a probability density function (PDF) and therefore it should be positive i.e.
$$f_r(p, \vec{w}, \vec{w'}) \geq 0 \tag{2.2}$$
2. **Helmholtz reciprocity**. Incoming and outgoing directions are considered to have the same reflection and swapping the directions in the BRDF function should not change the value. It can be formulated as :
$$f_r(p, \vec{w}, \vec{w'}) = f_r(p, \vec{w'}, \vec{w}) \tag{2.3}$$

3. **Energy conservation**. The energy of reflected light at the non-emitting surface point can not be higher then the energy of the incoming light. Thus, the following integral has to be less than 1 :

$$\int_{\mathcal{H}^2} f_r(p, \vec{w}', \vec{w}) \left|\vec{w}' \cdot \vec{n}\right| d\vec{w}' \leq 1 \tag{2.4}$$

There are several ways to derive BRDF: analytically, empirically or to measure in the laboratory. Measured BRDFs are represented as arrays of the reflectance and refraction at the different angles and positions. Some of the derived BRDFs for the computational simplicity do not comply to the physical constraints.

2.2 Physically Based Algorithms for Global Illumination

The amount of realism in the image rendered with algorithms based on the physics of light transport obtained big popularity in computer graphics.

2.2.1 Ray Tracing

Recursive ray tracing was introduced by Whitted [Whi79]. Ray tracing is a method of tracing of the rays, that originates at the camera and finishing at the light source. The algorithm can be described in the following way: First, primary rays are cast from the camera to the scene. Then the closest intersection point of each ray with the scene is calculated. If the ray does not intersect any object the background color is returned. If the intersection with the scene's object occurred, an intersection point for the nearest object is calculated. Afterwards, the shadow ray is cast to define if the object is occluded and is not lit directly by the light source. This part of computation accounts for the direct illumination. To account for indirect illumination besides the shadow ray two additional rays are cast: reflection ray and refraction ray. Their intensity is computed by recursively tracing the bouncing rays until the number of maximum depth is achieved or a light source is hit. Obtained results from these computations are scaled by Fresnel reflection and refraction probabilities and are summed up with the direct illumination and emitted light. The final result of light intensity in a specific direction is the sum of emitted, direct illumination and indirect illumination. It can be described as:

$$I_o(p, \vec{w}) = I_e(p, \vec{w}) + k_d \sum_{j=1}^{l_s} \left|\vec{w_j}' \cdot \vec{n}\right| + k_s S + k_t T \tag{2.5}$$

Where $I_o(p, \vec{w})$ is incoming intensity to the camera from intersection point, $I_e(p, \vec{w})$ is intensity of the emitted light, k_d is diffuse coefficient, $\vec{n}$ is normal of the surface at nearest intersection point, $\vec{w_j}'$ is ray towards j^{th} light source, S is intensity from recursive specular part, T is intensity from recursive transparent part, k_s and k_t are Fresnel's coefficients of reflectance and refraction respectively.

Originally the algorithm was targeting specular and transparent objects, because only one reflection and refraction angle was accounted complying to the Snell's law. Simulations with the use of such method could only produce sharp shadows, sharp reflections and sharp refractions. Later Cook [CPC84] introduced Distributed Ray Tracing. He proposed to distribute existing rays at the hit point in each dimension according to the parameters of the scenes. For example, to simulate the soft shadows it would require to distribute rays at the hit point in the direction of area lights. Distributed ray tracing can model such effects as glossy reflections, translucency, penumbras, depth of field and motion blur.

2.2.2 Monte Carlo Rendering

Monte Carlo rendering is the class of rendering algorithms that uses Monte Carlo integration method to approximate the integral in the rendering equation.

2.2.2.1 Monte Carlo Integration

Monte Carlo integration is general method for numerical integration. It utilizes integrand values taken at random points to evaluate integral with convergence rate that is independent of the dimensionality of the integrand. To evaluate a one-dimensional integral sampled by uniform random variables $X_i \in [a, b]$ we have:

$$\int_a^b f(x)dx \approx \frac{b-a}{N}\sum_{i=1}^{N} f(X_i)$$

The right part is called estimator and we will denote it as F_N. It is possible to show that the expected value of such estimator is equal to original integral. Random variable X_i is sampled uniformly therefore its PDF $p(x)$ is equal to $1/(b-a)$, according to the constraint, that $p(x)$ should be positive and integrate to one over domain $[a, b]$. Using the definition and additive property of the expected value:

$$\begin{aligned} E[F_N] &= E\left[\frac{b-a}{N}\sum_{i=1}^{N} f(X_i)\right] \\ &= \frac{b-a}{N}\sum_{i=1}^{N} E[f(X_i)] \\ &= \frac{b-a}{N}\sum_{i=1}^{N}\int_a^b f(x)p(x)dx \\ &= \frac{1}{N}\sum_{i=1}^{N}\int_a^b f(x)dx \\ &= \int_a^b f(x)dx \end{aligned} \tag{2.6}$$

It is possible to generalize the estimator for arbitrary PDF. This would give us the means to reduce the variance of the estimator. If random variable X_i has arbitrary PDF $p(x)$,

then the estimator

$$F'_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)} \tag{2.7}$$

can be used to estimate the original integral. To avoid division by zero we need to put a constraint on $p(x)$ to be non-zero for all x where $|f(x) > 0|$. Consequently the expected value of the estimator is :

$$\begin{aligned} E[F'_N] &= E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}\right] \\ &= \frac{1}{N}\sum_{i=1}^{N}\int_a^b \frac{f(x)}{p(x)}p(x)dx \\ &= \frac{1}{N}\sum_{i=1}^{N}\int_a^b f(x)dx \\ &= \int_a^b f(x)dx \end{aligned} \tag{2.8}$$

The uniform law of large numbers ensures us that the estimator at the infinity converges to the expected value. Thus, MC estimator equals to the original integral at infinity.

The Monte Carlo method does not pose restriction on the the integrand and the integral area. Only limited number of samples is needed for the evaluation of integral with certain precision. Besides that, the convergence rate of the method is invariant to the dimensionality, thus it is a good method for approximation of the rendering equation, which consists of high dimensional integral. The drawback of this method is its slow convergence rate $\mathcal{O}(\sqrt{N})$. With the small number of samples Monte Carlo estimator introduces approximation error in the amount of variance. The variance of MC estimator is proportional to the $\mathcal{O}(\frac{1}{\sqrt{N}})$.

2.2.2.2 Path Tracing

Path tracing was introduced by Kajia [Kaj86] as an approximation technique for rendering equation. Path tracing can be seen as the type of ray tracing algorithm with the main distinction in the way of casting secondary rays. In the path tracing secondary rays are cast multiple times in random directions according to the PDF of the MC estimator. When the light hits the surface the light integral is estimated with the MC estimator:

$$\int_{\mathcal{H}^2} f_r(p,\vec{w},\vec{w}')L_i(p,\vec{w'})\left|\vec{w}'\cdot\vec{n}\right|d\vec{w}' \approx \frac{1}{N}\sum_{j=1}^{N}\frac{f(p,\vec{w},\vec{w_j}')L_i(p,\vec{w_j}')\left|\vec{w_j}'\cdot\vec{n}\right|}{p(w_j)} \tag{2.9}$$

With the number of samples approaching to infinity MC estimator converges to the integral. Path tracing is a general rendering algorithm that can render broad variety of

the effects, such as area lighting, penumbras, depth of the field and others. To be able to render visually pleasant photorealistic image thousands of samples per pixel are required. With small number of samples the variance of the MC estimator is seen on the rendered picture as noise. Since the variance of the estimator is proportional to the $\frac{1}{\sqrt{N}}$ to reduce the amount of noise by two it is needed to take four times as much samples as were already taken. Variance of the estimator can be reduced with the Importance Sampling technique [Has70, Sie76]. Importance sampling is a variance reduction technique that exploits the fact that the Monte Carlo estimator (Equation 2.7) converges more quickly if the samples are taken from a distribution $p(x)$ that is similar to the function $f(x)$ in the integrand.

2.2.2.3 Bidirectional Path Tracing

Consider a scene where the light source is isolated at most of the directions, i.e. the scene is illuminated dominantly by the indirect illumination. Applying the path tracer to such scene will result in mainly dark scene. The reason for that is that only small amount of paths will end up at the light source. Bidirectional path tracing [LW93] was introduced to address this problem. The distinct novelty of bidirectional path tracing from simple path tracing is the use of forward propagation technique together with back propagation. To generate the path,two separate subpaths are generated, one originated at the camera and the other originated at light source. Afterwards, both subpaths are continued by sampling the directions from the BRDF function. Finally both of these subpaths form a single path by connecting the points of the subpaths if they are visible. Besides the better visual results for certain scenes the method has better convergence rate even with the computational overhead for maintaining two paths. The depiction of the algorithm can be seen in Figure 2.4

2.2.2.4 Russian Roulette

Russian roulette is a stochastic technique that helps to terminate recursive spawning of rays for indirect illumination. The biggest benefit of its utilization is achieved when the integrand has a small value, that means either attenuation is big or BRDF value is small. Algorithm consists of selecting ray termination probability q and termination of the path with this probability. The termination probability value can be chosen randomly or to have some intuition behind it, like estimate of the value of integrand that its contribution will be very low. With the probability $(1 - q)$ ray tracing continues but its result is weighted by $\frac{1}{1-q}$ that effectively account for all of the samples that were skipped:

$$F' = \begin{cases} \frac{F}{1-q} & \xi < q \\ 0 & otherwise \end{cases}$$

The expected value of the resulting estimator is the same as the expected value of the original estimator

$$E\left[F'\right] = (1-q)\left(\frac{E\left[F\right]}{1-q}\right) = E\left[F\right]$$

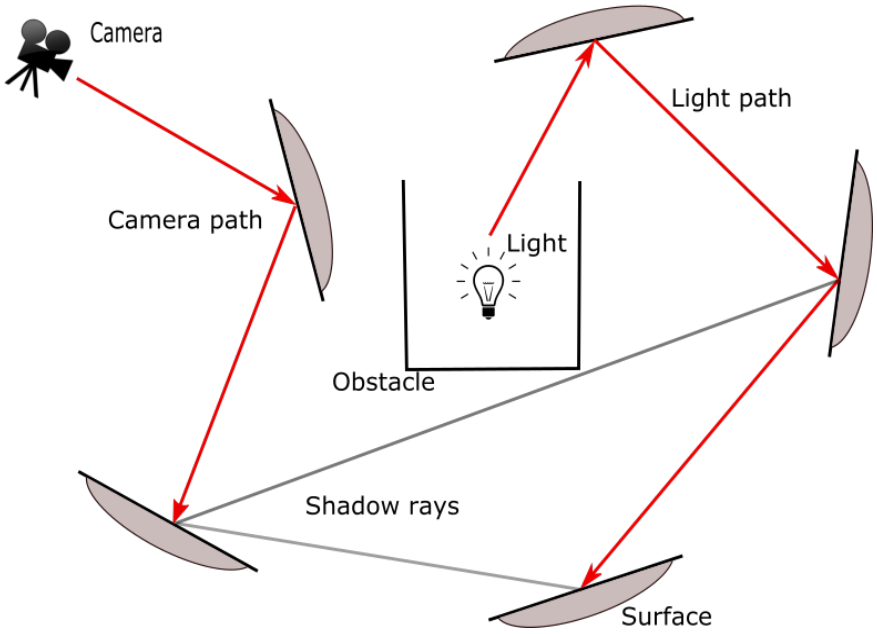


Figure 2.4: Bidirectional path tracing [LW93]. Two separate subpaths are traced originating in camera and the light source and later connected via the shadow ray.

and therefore it does not introduce bias. Russian roulette can only increase variance. But ingenious choice of the termination probability can significantly reduce computation time with having controllable increase in variance.

2.3 Filtering

MC rendering has slow convergence rate of $\mathcal{O}(\sqrt{N})$. It requires a lot of time to render visually pleasant photorealistic images. Without a high number of samples MC estimator introduces noise in the image caused by variance. One of the techniques that addresses the problem of the noise is image reconstruction from continuous light function (also known as filtering). Filtering technique uses information from the neighbour pixels and tries to filter out noise herewith preserving the features of the scene. Filtering equation can be described as :

$$f_i = \frac{\sum_{j=1}^{N} w_{ij} c_j}{\sum_{j=1}^{N} w_{ij}} \tag{2.10}$$

where N is number of pixels in the neighbourhood of the ith pixel, c_j is color value of jth pixel and w_{ij} are the weights, that signify the ratio of contribution of jth neighbour pixel to the filtered pixel. The main function of filtering represents substitution of the pixel value by the weighted sum of its neighbours. Pixel color could be either gray scale or in RGB form, in the later case c_j and f_j will be vectors with values (c_j^r, c_j^g, c_j^b) and

(f_j^r, f_j^g, f_j^b) respectively. The weights are obtained with the help of additional formula called filter's kernel.

One of the important properties of the filter is its consistency. The filter is called consistent if it converges to the ground truth with the number of samples tending to infinity.

2.3.1 General Image Denoising Algorithms

There are different filters' kernels, such as box filter, Mitchell, Sinc or Gaussian. Some of them have analytical basis, some of them were derived empirically. The most used filter in the signal transmission is Gaussian filter. In Gaussian filter the weights decrease exponentially with increasing distance between pixels. Its kernel can be formulated as :

$$w_{ij} = \frac{1}{\sigma\sqrt{2\pi}} \exp - \frac{\|p_i - pj\|^2}{2\sigma^2}$$

where p_i and p_j are the positions of the ith and j^{th} pixels respectively, σ^2 is a standard deviation of the Gaussian function and commonly used as the width of the square neighbourhood. After applying Gaussian filter to the noisy image together with the noise some of the image's features are blurred. The reason for that is the filtering based only on the spatial distance.

More specific filter, the bilateral filter [AW95, SB97, TM98] was proposed in the late 90s. Its main advantage is the ability to preserve edges. Tomasi and Smith [TM98] introduced extension of the Gaussian filter that takes into account scene's colors. Big difference in pixels colors signifies the discontinuity and therefore determines the edges and consequently decreases weight for two different pixels significantly, so that the pixels from the left side and the right side of the edge will be blurred and edge itself would be preserved. The simplicity and subtlety of this filter obtained popularity in the computer graphic community. One of the drawbacks of bilateral filter is slow filtering speed for the utilization in interactive systems. Several acceleration techniques were proposed to address this problem [DD02, PD09]. Chen *et al.* [CPD07] proposed an approach of edge aware filtering with bilateral grid. They divide the algorithm in three parts: splatting, blurring and slicing. During the splatting, bilateral grid is formed from the original image. Blurring is a process of filtering performed on the pixels inside bilateral grid. Conclusively, slicing means return to the image space by the use of trilinear interpolation. Adams *et al.* [ABD10] suggested permutohedral lattice as alternative for the bilateral grid and use of barycentric interpolation for splatting and slicing. Dammertz *et al.* [DSHL10] introduced the utilization of À-Trous wavelet transformation with edge stopping criteria to approximate cross bilateral filter using additional information including normals and world positions. He *et al.* proposed a method called guided filtering [HST10], that represents the filtered value as solution for minimization problem of a local linear model.

[BCM05] proposed extension and generalisation of the bilateral filter called Non-local means (NLM) filter. The novelty in their method is the utilization of the similarities of

pixel's patches trhoughout the whole image. The image has repeating patches of pixels that can give better information for filtering, since they contain multiple pixels. It is possible to derive bilateral filter from NLM by reducing the length of the patch to one pixel. The patches are compared trhoughout the whole image, hence a patch from one corner of the image can contribute to the patch of the pixels from the other corner. Thus the filter is non-local. NLM shows better quality of the filtered image comparing to the bilateral filter. The drawback of the NLM is that it is much slower and computationally expensive than bilateral filter. [MKSS14] proposed the acceleration method for NLM.

All these filters are Euclidean filters because of the use of Euclidean distance for distance computation between pixel's positions and colors. As alternative geodesic filters can be applied that use another metric for computation distance. The accelerated algorithms filter the images in the fraction of second, what permits their use in the real-time applications

[PSA+04] used combined information from the photos done with and without flash for filtering. They noticed that information from the image that was made with flash could be used to give additional information to filter the image that was made without flash. Such filter is called joint bilateral filter. Additionally, information obtained during the rendering such as normals, textures and world positions could be used to help to filter only the noise from the image and preserve actual details of the scene. The reason for that would be that this auxiliary information is less noisy than the color values. Filtering with the use of such features is called high-dimensional filtering. The fast high-dimensional filtering algorithm was presented by Gastal and Olivera [GO12]. Their algorithm uses manifolds as reduced dimension for splatting, recursive filter [GO11] for blurring and normalized convolution for slicing. A comprehensive overview on the state of the art can be seen in [SZR+15]

2.3.2 Filtering of Monte Carlo Noise with Adaptive Sampling

Hachisuka et al. [HJW+08] proposed a general multidimensional adaptive sampling algorithm. During adaptive sampling, more samples are put in the regions with high frequency in the multidimensional domain. Isotropic nearest neighbor interpolation was performed as reconstruction step. Algorithm becomes inefficient with increased dimensionality. In contrast to that, the work of Overbeck *et al.* [ODR09] is not prone to "curse of dimensionality". Overbeck proposed adaptive wavelet rendering algorithm for filtering and adaptive sampling. The algorithm distributes more samples to coarse scale coefficients in the smooth regions with high variance and to fine scale coefficients at the edges. Coarse scale coefficients and fine scale coefficients are the inner products between image and scale vawelet basis functions of most dilated level and the finest level respectively. The modification of a standard soft-thresholding denoising is used for reconstruction. However, at the low sampling rate (below 32 samples per pixel) the wavelet artifacts could appear. Several papers investigated frequency analyses for depth of field [SSD+09], motion blur [ETH+09] and soft shadows [EHDR11]. Rousselle et al. [RKZ11] proposed an approach for choosing the most appropriate filter for every

pixel with greedy algorithm that minimizes Mean Square Error (MSE). The drawback of the algorithm is restriction to isotropic filters. More recently, Rousselle et al. [RKZ12] introduced dual sample buffer for alleviation of the variance and error estimation. The utilization of Stein's Unbiased Risk Estimator (SURE) as an unbiased error estimator can be seen in [LWC12, RMZ13]. To cope with the noisy features [LWC12] divided the distance between features by sample variance and [RMZ13] suggested to pre-filter noisy features with NLM (Non-Local Means) filter. Kalantari *et al.* [KS13] presented a method for variance calculation with wavelet transformation and use of spatially-invariant denoising algorithms as BM3D or BLS-GSM for filtering. Moon *et al.*[MCY14] proposed the utilization of truncated single value decomposition (SVD) method for switching to reduced-dimensional coordinates, what also works as prefiltering for the noisy features. Additionally, local regression method was used for determination of the filters bandwidth.

2.3.3 Filtering of Monte Carlo Noise without Adaptive Sampling

Lehtinen *et al.* [LAC+11] use depth and motion information for reconstruction of the image from the samples to handle motion blur, depth of the field and soft shadows effects and global illumination [LALD12]. Sen *et al.* [SD12] presented Random Paramater Filtering (RPF) algorithm to compute functional dependency between features and random parameters and use obtained results to drive modified cross-bilateral filter to blur the noise and preserve scene details. They achieve high visual quality but the algorithm is computationally expensive and the memory consumption is excessive. In the recent work of Kalantari *et al.* [KBS15] neural network is used to find the dependency of the filter parameters on the secondary features, such as features statistics, gradients and mean deviation.

2.4 Genetic Programming

Genetic Programming (GP) is an optimization technique that mimics the principle of evolution - the survival of the fittest. This technique is general and domain-independent and can be applied to the vast types of problems. GP in a broad sense appears as a type of genetic algorithm but differs from it by operating on the executable structures, like programs represented in the Abstract Syntax Tree (AST). These structures are often represented in programming language, mathematical expression or boolean expression. There are variations of the way that structures could be represented such as linear or graph representation. In graph representation the structures are stored in the form of tree and evaluation is composed of recursive calls. In contrast to that in linear representation the form of instructions sequence is used. Good analogy for such form would be assembler code, that consists of stack of instructions.

The algorithm starts with forming the *population* of the programs that symbolize the potential solution of the defined problem. In each iteration every program in the population is assigned a fitness value, obtained by evaluation of the *fitness function*. Based on these fitness values some individuals of the population are *selected* to be the

parents for new generation. New generation is formed by *expansion* of the parents, i.e. modification of the existing programs. The algorithm is running until specified finishing criteria is met. The details of the algorithm are described below.

2.4.1 Program Representation

The programs are represented by the combination of the terminal and functional sets. The terminal set consists of user defined input (variables, vectors etc.) and functions with arity of zero, for example random function without parameters and constants. The functional set is a set of user defined functions that determines possible actions of the program. This set of functions is domain specific and should be devised properly. For example a simple arithmetic problem would consists of functions ($+$, $-$, $*$, $/$) and for the problem of labyrinth's escape the functions as **turn right, turn left, move forward** would be logical. Combination of these two sets form a trivial set of the programs. There are additional subjects that are needed to be considered during the definition of the trivial set: *closure* [Koz92] and *sufficiency*. Closure is set of constraints on the program represenation, that should hold after all possible modifications of program. Riccardo Poli and William B. Langdon[PLM08] partitioned closure into type consistency and expression safety. Type consistency is the constraint on the input parameters of functions. For example, some mathematical function should not have the input as a string. Expression safety persuades that no illegal operation occurs, like division by zero. Sufficiency is property of the trivial set that assures the ability of searched solution to be expressed by trivial set. For example we can not express solution of the searching approximation of logarithm function only with integers as terminals and the basic arithmetic functions.

2.4.2 Population Initialization

Population is the set of programs that serve as a source for possible solutions. Basically the solution is obtained from the initial population by sequence of the modifications. But this initial population should be defined before any modifications could be performed. There are several random techniques for creating such initial population: *full*, *grow* and *ramped half-and-half*. All of these techniques assume that maximum depth of the AST is defined. Maximum depth of AST is a restriction on maximum length of the program i.e number of nodes that is needed to be traversed from root till leaf (terminal value). In the *full* method the whole tree is generated randomly. The tree is balanced (all branches are of the same length specified by maximum depth value) with the functions as intermediate nodes and terminal nodes in the leaves. In contrast *grow* technique offers more variety on the shape of the tree. In the *grow* method the tree is formed in the following way. Initially, the function is randomly chosen for the root node. Following, we descend recursively by each edge of the parent node and randomly create new node, with some probability either from functional set or from terminal set and then select randomly an element of that set. If the maximum depth is reached we select randomly one element from the terminal set as the leaf node. Generated in such way, AST has different depths from leaves till the root. *ramped half-and-half* is a combination of these

two methods, such as the half of the population is generated by the full method and the other half by the grow method. In some of the works [ABI98, hChY97, FM95] the *seeding* is used to generate entire or to supplement initial population. Seeding is a process of initialization of population by the individuals of the same type. Taking into account the random nature of the algorithm's optimisation, good starting point can be as crucial as in the Newton's method. Sometimes *island model* [Gro85] is advisable for obtaining more diverse results. In island model the whole population is divided in small islands, which interact with each other seldom and only at specific occasions. Thus, the programs in each island interact only with the programs of the same island and the end result of the algorithm will be more diverse. The special occasions for interaction between islands could be a transportation of the best individuals to the other island after certain number of iterations. This technique is good for the parallel computation.

2.4.3 Termination Criterion

Termination criterion and method of the choosing the best solution among the population are important parts of Genetic Programming algorithm. Termination criterion is the criterion by achieving which we terminate our algorithm. There are two criteria for termination that are often used together: by reaching the maximum number of generations and by deriving and individual, that has better fitness value then predefined threshold. Maximum number of generations can be seen as a control parameter (see Section 2.4.4) because it influences the computational time. With the use of only maximum number of generations after achieving such number, individual or several individuals with highest fitness value are chosen as the solution. In case of threshold criterion the algorithm stops when it generates a program that satisfies the threshold value and takes it as a solution.

2.4.4 Selection

Selection denotes the process of choosing the parents for generation of new individuals by the modification operations. For the selection of parents the distinction between good and bad programs is required. This can be achieved by assigning the fitness value for every individual in population. Fitness value is a measure the programs' optimality according to the rigorously chosen fitness function. Fitness can be evaluated explicitly or implicitly. Implicitly fitness is evaluated when the individual survives the expansion. This method is more appropriate for the genetic algorithms in the life simulation. More common method of fitness evaluation in GP is explicit. In the explicit measuring of fitness every individual is assigned some scalar value that represents its fitness. It can be an amount of error between approximation and exact value of a function, the amount of time of the execution, the accuracy of the classification or relative payoff after playing with certain strategy against others. The evaluation of fitness almost always involves execution of the program. It consists of the recursive traversal of the AST tree and computing the results by backpropagating the computed values. After every individual is assigned the fitness value, certain selection methods can be applied. The most intuitive method for selection is proportional selection proposed by Holland [Hol75]. In the proportional selection the

individuals are chosen with the probability proportional to the their fitness values.

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j},$$

where p_i is the probability with which the ith individual from the population will be selected, f_i is its fitness value and N is the size of the population. As alternative to the proportional selection rank selection and tournament selection can be used. In rank selection [Bak89] every individual has been assigned a rank (integer value) according to its fitness value. Thus, the individual with the highest fitness value gets rank 1, the individual with second highest value rank 2 and so on. Afterwards the probabilities are distributed in the following way, described more detailed in [BT96] :

$$p_i = \frac{1}{N}\left(\eta^- + (\eta^+ - \eta^-)\frac{i-1}{N-1}\right); \quad i \in \{1, \ldots, N\} \tag{2.11}$$

where $\frac{\eta^-}{N}$ and $\frac{\eta^+}{N}$ are selection probabilities of the worst individual and the best individual respectively. η^+ is a selection parameter that can be chosen with the respect to the following constraints: $\eta^+ = 2 - \eta^-$ and $\eta^- \geq 0$. By performing such selection it possible to cope with the problem of domination of individuals with very high fitness relative to others. In the tournament selection the individuals are randomly selected into the group of specific size and then the individual with the highest fitness value is chosen. Normally the size of the tournament group is equal to two but can be varying according to demands of the problem.

2.4.5 Control Parameters Selection

Control parameters are the tuning parameters that can influence the end result of the algorithm. The most important control parameter is the size of the population. There is no best suitable number for every case, this parameter is problem dependant. The algorithm's computation time is dependent on the size of the population. The bigger the size of the population the more fitness evaluations and expansion operations should be performed. In the majority of the GP algorithms the assignment of the fitness value is a bottleneck. The most common size of population is 500. As it was mentioned in Section 2.4.3 maxmium number of generations is also a control parameter. Since every iteration contains fitness evalutation of every individual in population, the computational time increases linearly with the increase of maximum number of generations. Another parameter that have an impact on computational time is the maximum depth of AST representation of program. The bigger the program is the longer it takes to evaluate it. Besides that there are two selection parameters of the algorithm. The first one is a ratio of expansion operation, wich decides how the new generation is formed. Koza [Koz92] suggests to use 90% of crossovers without mutations. Also other ratio can be used to form new generation that proved to be efficient, such as selecting half of population and perfroming crossover on them and forming other half by taking unmodified individuals with applying mutations on some of them. The second selection parameter

is the probability density function for selecting the nodes in the AST for crossover and mutations. Uniform probabilty can be used as an example or as suggested by [Koz92] choose with 90% probability from the functional nodes and with 10% probability from the terminal nodes.

2.4.6 Expansion

In order to get the solution for the problem from the initial population, modification (alternation) of the individuals is needed. Commonly used alternation operations in GP are crossover and mutation. The crossover (recombination) is an operation of breeding children from selected individuals. Crossover operation needs two individuals to act as parents and therefore two selections from the population should be performed. The most popular crossover variant used in GP is subtree crossover. Subtree crossover can be described in following way: Initially, crossover points (nodes in the AST trees) are selected with a certain probability from both parents. Then a new AST is obtained by swapping the selected nodes between two trees and selecting the first tree. Herewith the whole subtrees that start at the selected nodes are swapped. Mutation is an operation in which a new individual is produced by the alternation of the existing individuals. The forms of mutations that are the most commonly used in GP are subtree mutation and point mutation. In subtree mutation a mutation point is selected with the same probability as the crossover point and a subtree rooting in that point is substituted by randomly generated tree. Opposite to subtree mutation, in point mutation only the selected node is replaced. The selected node is replaced with equivalent node, which is of the same type (functional, terminal) and if it is the functional type with the same arity. More forms of the crossover and mutation can be found in [PLM08]. Besides these two operations a worth noting one is reproduction, in which the individual from current generation is placed in the next generation unchanged. Reproduction is used as supplementary operation to form lacking number of individuals to fill the whole population of new generation, after crossover and mutation are performed. The description of another domain specific alternation operations such as permutation, editing, encapsulation and decimation can be found [Koz92].

2.5 Genetic Programming in Computer Graphics

Although GP has a lot of applications in different areas it still have not gained popularity in the computer graphics. Worth notable works on Genetic Programming in the computer graphics are [SAMWL11] for shader simplification and [BLPW14] for finding novel BRDF formulas. We have have taken an inspiration in the work of Brady *et al.* [BLPW14] and investigated applicability of the GP algorithm to the filtering problem.

CHAPTER 3

Generation of New Filtering Expressions with Genetic Programming

This chapter describes the details of our GP algorithm for finding novel filtering expressions. Section 3.1 shows overview of the search for new filtering expressions, Section 3.2 describes the algorithm and Section 3.3 describes the implementation details, such as technology stack, generation of input data and decisions made in order to increase performance.

3.1 Overview of the Search Algorithm

The process of searching for new filtering expressions can be visually represented in diagram shown in Figure 3.1 and described in following way: First the input for the algorithm is generated. The input is the representation of the training scenes with additional parameters for the algorithm. Then this input is parsed by the program. After processing of the input is finished the decision on the parameters of the algorithm has to be made. Another operation, that should be performed before the start of the algorithm, is formation of the initial population. The iterative part of the algorithm consist of:

- selection - the process of individuals selection in the population to act as the parents for the new generation. The individuals are selected according appropriate selection method (see Section 2.4.4).
- expansion - the process of alternation of existing individuals to explore the solution space. Expansion operators (see Section 3.2.4) are applied to the selected individuals with specified probabilities.

- fitness assignment - the process of evaluation of the expressions. After the new generation is formed a fitness value should be assigned to each expression. This fitness value helps to distinguish the individuals during the selection phase in the next iteration.

Finally, as the termination criterion is met, the iterative part is terminated. The result of the algorithm is obtained by sorting the individuals of the latest generation by fitness value and selecting the best ones. The best representatives are then checked manually for visual pleasantness of the results. Each part is described in detail in the following sections.

3.2 Genetic Programming for Filtering

3.2.1 Operators

In order to modify a filtering expression we need to represent it in machine-understandable way. As described in the Section 2.4.1 the trivial set, consisting of functional operators and terminal values, should be defined. To get the idea of how to define the operators already existing filtering expressions can be helpful. Extracting operators and values out of them will create initial set. Having in mind the *sufficiency* condition we need to think of other possible operators that could improve the expression's fitness value. The *closure* constraint does not hold in strict sense in our case, because there are feature vectors that have higher dimensions and can not be used as an input for the operators beside vector operators. We represent the trivial set in form of grammar, so that only specified operations are allowed for functional and terminal sets. This grammar can be seen in Figure 3.2 .

3.2.1.1 Functional Operators

The set of functional operators consists of unary, binary and vector operators.

Unary Operators To aspire the diversity we use trigonometric functions to introduce non-linear dependency on the parameters. Operators of exponent and negation are taken from the Gaussian filter. Even tough the square root can be represented with binary operator power as $sqrt = pow(x, 0.5)$, square root operation is faster. Besides that several statistics kernels are used that are similar to Gaussian kernel and operators from the already existing filters, that help to form the codebook (see Section 3.2.2), such as : mitchell, sinc, epanechnikov, tricube and biweight kernels.

Binary Operators Main binary operators are arithmetic operations between two numbers, which include sum, subtraction, multiplication and division. In order to comply to type safety we use safe division function to avoid division by zero. Other binary operator is power. Power is an exponential function that takes first parameter as a base and the second parameter as the exponent.

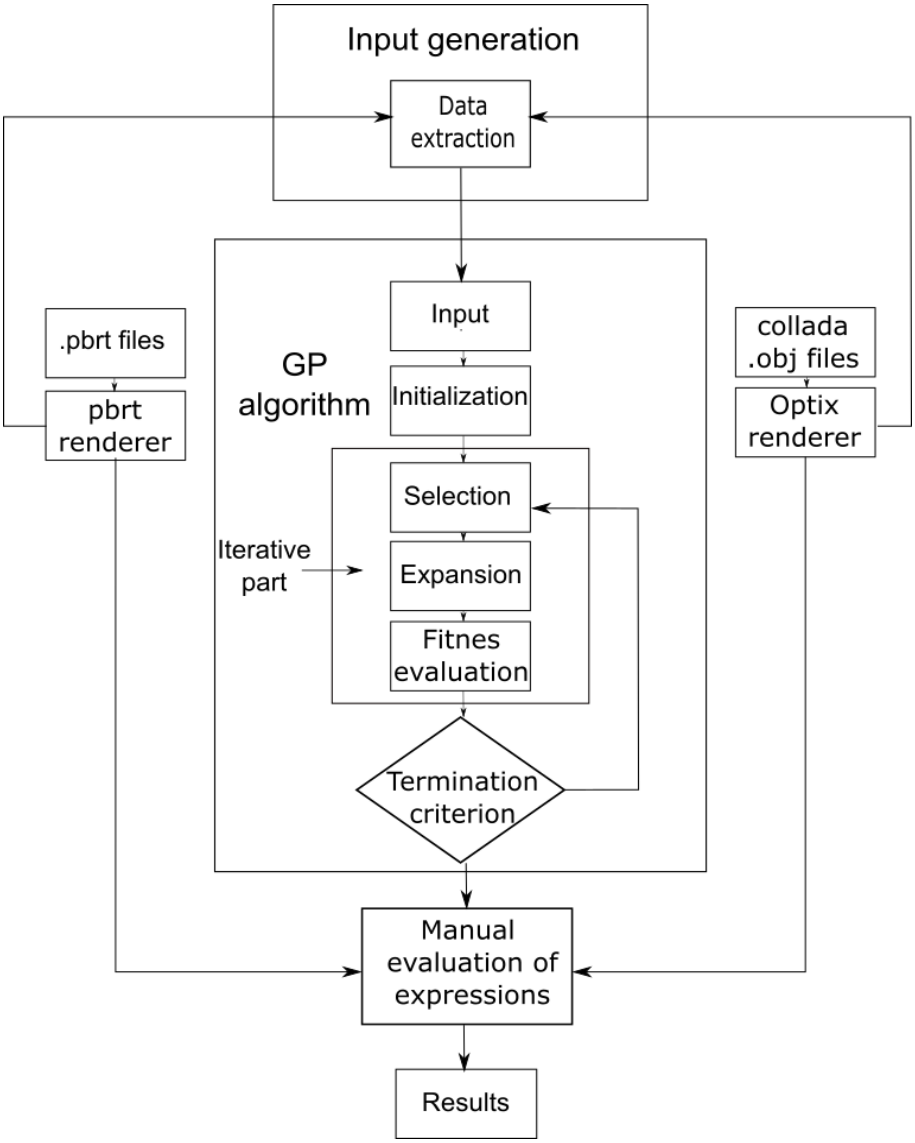


Figure 3.1: Overview of our search for new filtering expressions.

Vector Operators The set of vector operators consists of dot product and different distance measuring metrics. The motivation behind the inclusion of dot product is its naturality of utilization between two normals. We have three distance metrics between two vectors $x(x_1, x_2 \dots x_N)$ and $y(y_1, y_2 \dots x_N)$ where N is dimensionality of the vectors:

- *distance2* - is an Euclidean distance

$$d = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2}$$

$$
\begin{aligned}
\langle expression \rangle &::= \langle node \rangle \\
\langle node \rangle &::= \langle op \rangle \,|\langle scalar \rangle \\
\langle op \rangle &::= \langle unaryOp \rangle \,(\langle node \rangle) \\
&\quad |\langle binaryO \rangle \,(\langle node \rangle \,, \langle node \rangle) \\
&\quad |\langle vectorOp \rangle \,(\langle vector \rangle \,, \langle vector \rangle) \\
\langle unaryOp \rangle &::= \; - \mid sin \mid cos \mid tan \mid exp \mid asin \mid acos \mid atan \mid sqrt \\
&\quad \mid mitchell \mid sinc \mid epanechnikov \mid biweight \mid tricube \\
\langle binaryOp \rangle &::= \; * \mid - \mid + \mid / \mid pow \\
\langle vectorOp \rangle &::= \; dot \mid distance2 \mid distance1 \mid distanceMax \\
\langle vector \rangle &::= \; worldPosition \mid normal \mid texture \mid secondaryTexture \\
&\quad \mid depth \mid directIllumination \mid wpGradient \mid nGradient \mid texGradient \\
&\quad \mid secTexGradient \mid dovGradient \mid diGradient \\
\langle scalar \rangle &::= \langle variable \rangle \;\; |\langle const \rangle \\
\langle variable \rangle &::= wpVariance \mid nVariance \mid texVariance \mid secTexVariance \\
&\quad \mid dovVariance \mid diVariance \mid wpGVariance \mid nGVariance \mid texGVariance \\
&\quad \mid secTexGVariance \mid dovGVariance \mid diGVariance \\
\langle const \rangle &::= \; 0.1 \mid 0.2 \mid 0.3 \mid 0.3333 \mid 1.0 \mid 2.0 \mid 3.0 \mid \pi \mid 5.0 \mid 7.0 \mid 11.0 \mid 13.0
\end{aligned}
$$

Figure 3.2: The grammar used for generation of novel expressions.

- *distance1* - is a Manhattan distance

$$d = \sum_{i=1}^{N} |x_i - y_i|$$

- *distanceMax* - is a Chebychev distance

$$d = \max_{1 \leq i \leq N} |x_i - y_i|$$

3.2.1.2 Terminal Variables

Terminal variables serve as an input for the functional operators. Terminals variables consist of vector variables, scalar variables and constants.

Vector Variables The set of vectors consists of auxiliary features that were extracted during the rendering of the scene. These features are less noisy and can give valuable information for determination of the noisy pixels. Values of these features are determined

for every pixel in the scene. To determine these values at the arbitrary coordinates (x, y) a ray is shot from the camera eye position in the direction of the coordinates. If the ray intersects an object we store the following data:

- *world position* - spatial three dimensional coordinates of ray's intersection point with the object in the world coordinate system.
- *normal* - normal of the surface of that object at the intersection position.
- *texture* - object's texture value at the position of intersection.
- *depth* - the distance between intersection point and the camera position.

Valuable information for the glossy surfaces is *secondary texture*, because it is possible to see the reflection of other objects on it. Secondary texture is a value of the texture at the intersection point of the reflected ray. As proposed by [RKZ12] *direct illumination visibility* can be used as additional information. Direct illumination visibility is a fraction of light sources, visible at the intersection point. Gradients are proved to be good indicator of the edges [KMA+15, MVZ16], thus we compute gradients for most of the features by Sobol operator. The auxiliary feature *depth* is calculated as distance between *world position* and camera position. Therefore depth gradient values are dependent on world position gradient values and we decided not to include them. The example of how features look can be seen in Figure 3.3.

Scalar Variables Scalar variables consist of the sample variance of the features. The variance values of depth are not included in the set for the same reason as depth gradients were not included in the features set. The color's sample variance is one of the most valuable components for error estimation as can be seen in [RKZ11, LWC12]. The desirable expectation from the variables is to be related to the filter parameters for different features, like the the σ in the Gaussian filter.

Constants Constants play important role for the modification of the expressions. It is the simplest way to tune the expression by multiplying or dividing by a constant value. For the replace mutation (which will be described later in Section 3.2.4) both variables and constants are chosen uniformly. To avoid the dominance of choosing constants over variables we have restriction on the number of constants. Which constants were chosen to be representative can be seen in Figure 3.2.

3.2.2 Codebook

Codebook is union of the sub-expressions of the predefined analytical expressions. We populate the codebook with:

- *mitchell* is a kernel of Mitchell-Netravali filter [MN88] (also known as BC-splines). This kernel can be formulated as

$$k(x) = \frac{1}{6}\begin{cases} (12 - 9B - 6C)\,|x|^3 + & if\ |x| \leq 1 \\ (-18 + 12B + 6C)\,|x|^2 + (6 - 2B) & \\ (-B - 6C)\,|x|^3 + (6B + 30C)\,|x|^2 + & if\ 1 \leq |x| \leq 2 \\ (-12B - 48C)\,|x| + (8B + 24C) & \\ 0 & otherwise \end{cases} \tag{3.1}$$

  This is an empirically derived kernel. It has two tuning parameters B and C. The subjective result of combinations of different parameters values can be seen in the Figure 3.4

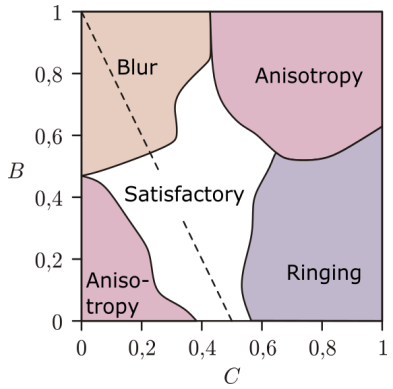


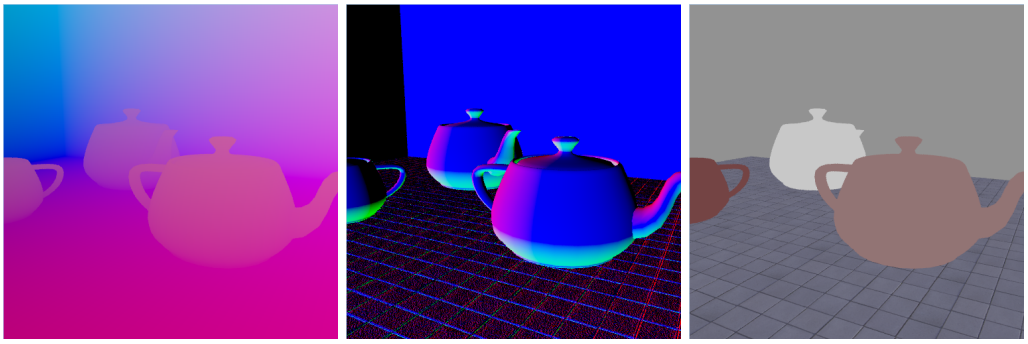
Figure 3.4: Subjective look of the filtered picture with different tuning parameters. Image courtesy of Don Mitchell [MN88].

  The default value for both B and C is $1/3$

- *sinc* is a low-pass filter used in the signal processing. Its kernel can be described as

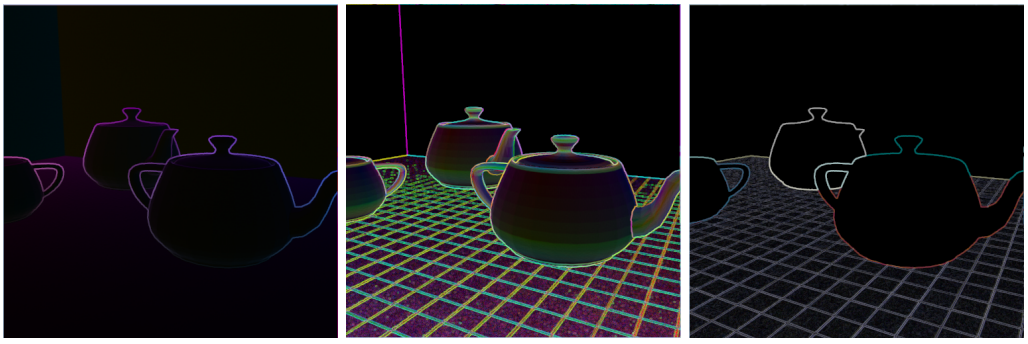
$$k(x) = \begin{cases} 1 & if\ x = 0 \\ \frac{\sin(\pi x)\sin(\tau \pi x)}{\pi^2 x^2} & if\ 0 \leq |x| \leq 1 \\ 0 & if\ |x| > 1 \end{cases}$$

  where τ is a tuning parameter that determines the size of the kernel and its default value is 3.0



(a) World positions (b) Normals (c) Textures

(d) Secondary textures(ST) (e) Depth (f) Direct illuminations(DI)



(g) World positions' gradients (h) Normals' gradients (i) Textures' gradients

(j) ST' gradients (k) Depths' gradients (l) DI' gradients

Figure 3.3: Features representation. All the features are mapped to the interval of $[0, 1]$.

- *epanecnhikov* - is a kernel originally introduced by Epanechnikov [Epa69] for estimation of multivariate probability density. Its kernel is

$$k(x) = \begin{cases} \frac{2}{\pi}\left(1 - |x|^2\right) & if\ |x| < 1 \\ 0 & otherwise \end{cases}$$

- *biweight* is a statistical symmetrical kernel, that can be described as

$$k(x) = \begin{cases} \frac{15}{16}\left(1 - |x|^2\right)^2 & if\ |x| \leq 1 \\ 0 & otherwise \end{cases} \tag{3.2}$$

- *tricube* is another statistical kernel

$$k(x) = \begin{cases} \left(1 - |x|^3\right)^3 & if\ |x| < 1 \\ 0 & otherwise \end{cases} \tag{3.3}$$

- variation of Gaussian filter, where distance between pixel position is substituted by Euclidean distance between features with the σ of 0.1

The codebook generation process can be described by the following example. Lets assume that we want to populate codebook with Gaussian filter for world positions

$$e^{-\frac{||f_i^{wp} - f_j^{wp}||^2}{2 \cdot 0.1^2}}$$

then the codebook would consists of all sub-expressions of the filter

$$\begin{aligned} codebook = \{ & e^{-\frac{||f_i^{wp} - f_j^{wp}||^2}{2 \cdot 0.1^2}},\ -\frac{||f_i^{wp} - f_j^{wp}||^2}{2 \cdot 0.1^2},\ \frac{||f_i^{wp} - f_j^{wp}||^2}{2 \cdot 0.1^2},\ ||f_i^{wp} - f_j^{wp}||^2, \\ & 2 \cdot 0.1^2,\ 2 \cdot 0.1, \cdot 0.1^2,\ f^{wp},\ 2,\ 0.1\} \end{aligned}$$

for the simplification reasons the square of the distance is computed inside the distance function to avoid the creation of additional node in the tree.

3.2.3 Fitness Function

To be able to steer our algorithm to the fittest solution, we need to distinguish bad expressions from the good ones. For that reason fitness function is essential for the GP algorithms. If fitness function can not fully measure how good the expression is, then it will not guide the algorithm to the best possible solution. In the computer graphics it is common to use mean squared error (MSE) for the error estimation. MSE uses averaged squared difference between the ground truth color and noisy color :

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \sum_{q \in \{r,g,b\}} (\hat{c}_{i,q} - g_{i,q})^2 \tag{3.4}$$

where N is the number of pixels, $\hat{c}_{i,q}$ and $g_{i,q}$ are i^{th} pixels and q^{th} channel of filtered color value and ground truth color value. We are going to use relative MSE (relMSE) proposed by Roussele *et al.* [RKZ11] and slightly modified by Kalantari [KBS15]:

$$relMSE = \frac{1}{N}\sum_{i=1}^{N}\frac{m}{2}\sum_{q\in\{r,g,b\}}\frac{(\hat{c}_{i,q}-g_{i,q})^2}{g_{i,q}^2+\varepsilon} \tag{3.5}$$

where m is the number of samples per pixel and ε is small value (0.01 in our implementation) so that division by zero will not occur. Division by ground truth value in formula accounts for the fact that human vision is more sensitive to darker regions. Multiplication by half of the number of samples per pixel prevents the introduction of the bias by preferring expressions with high number of samples and small error. The final fitness value of the expression is calculated as averaged sum of relMSEs computed for every training scene. It could be the case that with the same number of pixels scenes with fewer details would have less noise than some more complex scenes. The difference in the initial relMSE, i.e. relMSE computed for noisy colors, can be significant between such scenes. The algorithm can be biased in favour of expressions that filter complex scene better, since it has bigger error minimization. To address that problem we normalize every relMSE obtained from the scene by dividing it by initial relMSE.

Some of the functional operators have limited domain range, for example square root in the field of real numbers can be extracted only from positive numbers. To assure the type safety property if the undefined operation occurs the fitness value is set for such expressions to a very big number (maximum float in our implementation).

3.2.4 Expansion Operators

Crossover, mutation and replication are typical expansion operators used in Genetic Algorithms. The crossover is pairing operation that provides two new individuals by recombining sub-expressions between two individuals from the current generation that were chosen to act as parents. It is the most important operator in the GP and it produces the most diversified expressions. Adapted to our problem, crossover can be described in the following way. Initially we select two individuals to act as the parents. After the selection is done the crossover points (specific nodes in the AST) should be defined for every parent. If the chosen nodes are interchangeable their whole sub-trees are swapped. Interchangeable in this case means that the types of the nodes are compatible, either both of them are vector operators and vector nodes or scalar operators and scalar nodes. If the nodes are not interchangeable then the process of selecting crossover points is repeated until they are interchangeable.

We use two types of mutation operators: point mutations and sub-tree mutations (see Section 2.4.6). For all mutations we select an individual for a mutation. The only point mutation technique that we use is *replace*. In the replace operation a mutation point is selected according to the selection probability (3.2.6) and only this node is replaced, leaving children nodes unchanged, by the equivalent node from our grammar. The

examples of equivalent replacement are: binary operator is replaced by binary operator, vector operator is replaced by the vector operator. It is possible to replace constant nodes with variables and other way around. Sub-tree mutations that we use are:

- *swap* - Swap operation can be seen as the crossover operation performed only on one tree. By the swap operation two mutation points are selected and checked for the interchangeability. If they are interchangeable the whole sub-trees rooted at the mutation points are swapped. Otherwise selection of nodes is repeated until it is possible to swap.

- *insert* - In the insert operation the sub-tree rooted in selected node is substituted by the compatible sub-expression from the codebook.

- *delete* - In the delete operation the sub-tree with the root in mutation point is replaced by the constant node of one. If the selected node is vector node we go up by the tree to vector operator and replace that operator with the constant node.

The mutation operators were inspired by the article of generating new BRDF functions [BLPW14]. In the replication operation the individual of current generation is moved to next generation unchanged.

3.2.5 Initial Population

To look for more diversified results than for just minor alternation of initial population the island model [Gro85] is used. In the island model the whole population is divided into isolated sub-populations that do not interact with individuals outside their islands. To profit from the isolated populations occasionally the migration of the best representative is performed. We used the migration strategy used by Brady [BLPW14] for performing such operation. Every 5th generation the best representative of each island is migrated to the neighbour island. Thus, if we are in the island i we will migrate the best representative of this island to the island $(i + 1)$ *mod* n where n is the number of the islands in the population.

Although it is common to use randomized tree generation techniques as full, grow and ramped half-and-half for our case we decided to use only seeding for the whole population initialization. As a basis for the initial expressions we chose the bilateral filter. Bilateral filter has relatively short formula and is proven to work well in practice. If the unmodified bilateral filter is used for initial population the crossover operations would be reduced to the replace operation for the first iteration. So the initial population is formed in the following way. We fill 10 percent of the island with the bilateral filter and form the rest by performing mutations on the bilateral filter. Such operation is performed to fill every island.

3.2.6 Parameters Selection

Important parameters to be set before GP execution are the number of iterations and the size of the population. Hastily assigning big numbers to these parameters is computationally expensive. In other hand if the number of generations is small there is small probability that algorithm will derive good expressions and small size of population hinder the diversity of choice. Due to the number of population is specially influential on performance it was derived empirically that number of 400 expressions is enough. These 400 expressions are partitioned to 4 different islands. The number of iterations was set to 200, since the results with fewer iterations were not satisfactory and with bigger number of iterations the running time of the algorithm is too big for running sufficient number of experiments.

Regarding the selection probabilities following decisions were made. At each generation a new sub-population is derived from previous by using 10 % of replications, i.e. unchanged individuals are propagated to new generation, 45 % of nodes are generated from crossovers and 45% of nodes are generated from mutations. Additionally mutation is performed with 0.85 probability on the crossover children. This decision was made in order to strive for diversity and minimising possibility to stuck in the local minimum. Each individual is selected by the tournament selection method with the group size of 8. During this tournament selection 8 individuals are selected uniformly from the population and the winner, the individual with the lowest fitness value, is used as a parent for generation of new expression. For the selection of crossover and mutation points the uniform distribution was used. Opposing to the suggestion of selecting 90 % of nodes with functional terms and 10 % with terminal term the uniform distribution was chosen. The reason for that is that the terminal terms, represented by variables and constants, can be seen as tuning parameters for the features, what is essential for the good work of the filter. For the vector terminal term it is comparably important because the presence or absence of certain feature in the expression can have crucial impact on the result. For the selection of the mutation types following distribution is used: 30 % for insert, 30 % for replace, 30 % for swap and 10 % for delete. Uniform distribution is also used for the the sub-expression selection from the codebook and equivalent node selection for the replace operation. The reason for that is unknown importance of different selections. Therefore, the same weights are preferable.

We used the maximum number of iterations as the ending criterion. The utilization of threshold criterion for the algorithm termination is problematic since the fitness is average of scenes' relMSE and every scene is different in the complexity and amount of noise. Top expressions from every island are chosen as results after sorting the individuals according to their fitness values. Finally selected expressions are manually checked on testing scenes.Because of performance issues described in the Section 3.3, the restriction on the length of the expression was set in our implementation to 75.

3.2.7 Input

Additional input to our algorithm includes:

- resolution of the scene
- number of samples per pixel
- number of scenes
- filter width
- noisy values - noisy color values for every pixel in rgb
- ground truth values - color values for every pixel in rgb rendered with either 32768 or 98304 samples per pixel
- 3D features - 3 dimensional feature values for world positions, normals, textures, secondary textures and their gradients for every pixel
- 1D features - 1 dimensional feature values for depth, direct illumination and their gradients except for depth
- sample variance - sample variance for features except depth

3.3 Implementation

In this section we describe the technology stack of used framework and implementation details. The main emphasis is made on GPU computation as it takes the most execution time of the algorithm.

3.3.1 Hardware

The GP algorithm was executed on the PC with Intel(R) Core(TM) i7-4790K CPU 4.00G Hz with 32.0 GB RAM and Graphic Card Geforce GTX 980Ti. The minimum requirement on the hardware is support of NVIDIA CUDA[1] on the GPU. If there is a demand for the generation of input for our algorithm the requirement of a machine to be 64-bit rises.

3.3.2 Input Generation

For the rendering of the scenes we used Optix [PBD+10] version 3.8.0 and PBRT [PH10] version 2. Optix is a general purpose ray tracing engine from NVIDIA. As the basis a simple path tracer program from Optix SDK is used. PBRT is a physically based renderer that can render a wide range of effects, such as depth of field, motion blur,

[1] www.nvidia.com

(a) Plants godrays scene (b) San Miguel scene (c) San Miguel scene

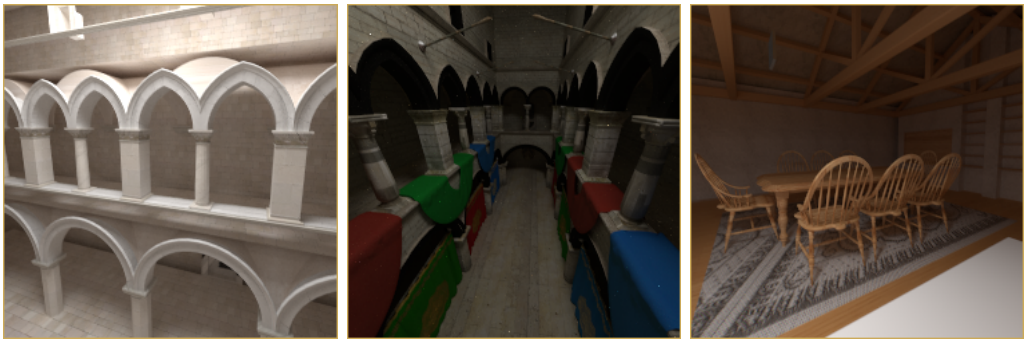
Figure 3.5: Training scenes obtained via PBRT renderer. Plants godrays scene is the courtesy of Oliver Deussen, San Miguel scene is the courtesy of Guillermo M. Leal Llaguno.

participating media, area light sources, etc. There are two ways of generating the input : by rendering scenes using PBRT or Optix.

Extracting features from PBRT. For rendering the ground truth data PBRT version 2 was used. The scenes were downloaded from the `www.pbrt.org\scenes`. To generate noisy data and extract features the implementation of Kalantari *et al.*[KBS15] was used. In order to sync with the implementation of Kalantari during the rendering of ground truth, all of the scenes were rendered with discrepancy sampling and photon mapping with metropolis sampling was discarded . There was not a straightforward way to extract depth informations from PBRT rendering so it was calculated by taking Euclidean distance between world position and camera position. The gradients of the features were calculated with Sobol operator. The training scenes of our method, obtained with PBRT, can be seen in Figure 3.5.

Extracting Features from Optix Our implementation supports rendering of the scenes represented in collada [AB06] and obj files. The scenes were obtained from the public repositories. Blender[2] and 3D Max [3] was used for editing the scenes. For importing scene's description from collada and obj file formats part of the implementation of [Ká14] was used. In that implementation, assimp[4] library was included for parsing the files. The features are extracted from several programs of the rendering pipeline. The rendering pipeline consists of *ray generation program*, *intersection program*, *closest hit program*, *any hit program* and *miss program*. The features are extracted from intersection program, closest hit program and any hit program. Intersection program computes the nearest intersection point of the emitted ray with scene. Closest hit program implements the shading and extracting the colors from the intersected object's material. Any hit program

[2] `www.blender.org`
[3] `www.autodesk.com//products//3ds-max//overview`
[4] `http://www.assimp.org/`



(a) Dabrovic scene (b) Crytek scene (c) Interior scene

Figure 3.6: Training scenes obtained via Optix renderer. Dabrovic scene is the courtesy of Marko Dabrovic, Crytek scene is the courtesy of Frank Meinl and Interior scene is the courtesy of Giiman from the www.turbosquid.com.

checks the visibility of the ray by the light source and returns true if the light is visible and false otherwise.

World positions are extracted in intersection program by using the intersection point. Depth was calculated by taking distance between intersection point and camera position. Normals are also calculated in the intersection program. Texture values are extracted at intersection point in the closest hit program by requesting the materials texture value at that point. Secondary texture is extracted the same way as primary texture at the intersection of second bounce. Direct illumination is calculated as an average of the visibility values obtained from shadow rays.

All of the feature values including the gradients are normalized. One dimensional feature values were mapped to the interval between $[0, 1]$ by dividing all values by the maximum value in the scene. Three dimensional feature values were normalized by dividing component-wise each value of the feature by the length of the biggest vector. This is done to ensure the interchangeability between features during expansion processes. The training scenes obtained with Optix can be seen in Figure 3.6.

3.3.3 Representation of Abstract Syntax Tree

Expressions are represented in the AST tree. This tree is represented as linked list with the references to the parent node and children if they exist. Different types of nodes are implemented in the following way. There is a base class that every specific type of node inherits from. There are 6 derivative classes: binary operator node, unary operator node, vector operator node, vector node, variable node and constant node. The value of the expression is evaluated in the recursive way with input parameters of scenes number and coordinates of center pixel and neighbour pixels. For the sake of performance optimization evaluation of fitness function was implemented on GPU using CUDA version 7.0. For the GPU implementation some work-around was needed. GPU and CPU have different

memory address spaces. Therefore, there is a need to transfer data from the RAM to the graphics card. The most important data to transfer is the set of newly obtained expressions for evaluation. AST tree is represented as linked list and every node has pointers to the child and parent. This pointers are represented as memory addresses in the CPU and it is not possible to transfer the whole tree, because the memory addresses are different and pointers will point to the wrong memory in the GPU. Another problem with the AST tree is that evaluation of expression requires a recursive run from the root of the tree to the leaves. Even though CUDA supports recursion from version 4.0 it is very slow. The solution to such problem is utilization of the stack. Before the GPU fitness evaluation every expression is transitioned to the stack and loaded to the graphic card. Besides the expressions, features and variables should be also loaded to the GPU. This is done using textures. Before we can use Equation 3.2.3 to compute the weights unwinding of the stack should be performed. The computation of the weights is the bottleneck of the implementation. Nevertheless, as can be seen in the Section 4.1.1 the boost of performance is significant by utilizing GPU.

3.3.4 Evaluation of Expressions

The final results, obtained from our optimization, are the filtering expressions sorted by their fitness values. It is possible that one filtering expression with higher numerical error gives more visually satisfactory image than the expression with lower error. For that reason manual evaluation of the potential best expressions is needed. The resulting set of expressions is restricted to the top 10 expressions from every island due to the time consuming manual evaluation. Such manual evaluation is done by rendering the datasets and testing scenes at 4 samples per pixel, then filtering the image with a given expression and measuring the results. These measurements consist of relMSE for the specific scene, filtering time and subjective visual pleasantness ranging from 0 to 10. For the evaluation of the expressions Optix framework is used. Filtering function with obtained expressions is implemented as additional ray generation program. This ray generation program computes for every pixel the importance weights of the neighbours with filtering expression and substitute the pixel color by the weighted average of the neighbours.

CHAPTER 4

Results

This chapter evaluates the main results of the thesis. We perform time measurements of our algorithm and observe influence of certain parameters in Section 4.1. Besides the performance, relative error is measured and discussed in Section 4.2. Additionally to that, we answer the third research question in Section 4.3. Finally, we evaluate our generated expressions in Section 4.4 and assess the whole algorithm in Section 4.5

4.1 Performance Evaluation

4.1.1 CPU vs GPU

GP algorithms are generally demanding big computational power or computational time. The straightforward implementation for computation of fitness values on CPU becomes inapplicable for the resolutions of 512x512 and more. In order to speed up the algorithm and fit the computational time into acceptable 2-3 days the GPU computation was used. Computation of fitness values was performed on GPU with the use of CUDA and data was transferred from CPU to GPU as textures. With the use of GPU computation the tangible speed up was achieved, that can be seen in the Table 4.1.

Resolution in pixels	**GPU** [s]	**CPU** [s]
32x32	0.457	2.667
128x128	6.799	734.623
256x256	22.038	3165.72
512x512	183.14	27501.9

Table 4.1: Comparison of the running time of one iteration on GPU and CPU with different resolutions.

4.1.2 Perfromance Measurements

In our evaluation we measured the runtime of our Genetic Programming algorithm with different sizes of population, maximum number of iterations and resolutions. As long as it is not mentioned explicitly all measures are made with the population of 4 islands, each with 50 individuals, number of iterations of 50, resolution of 512x512 and the teapots scene shown in Figure 1.1b as training scene. Several insights can be derived from the performed measurements:

As can be seen in Figures 4.1, 4.2, 4.3 the running time is increasing with increase of population size, number of iterations and resolution. There is no straightforward linear dependency on the length of the expressions, what can be seen in Figure 4.4. The reason for that would be that the time spend on the fitness assignment depends a lot on the number of requests to the global memory and such requests are much slower than requests to local memory. Reading feature values is a request to global memory. Therefore, smaller expression with high number of feature distance calculation can have higher computational time then bigger expression, that contains mostly arithmetic operators with constants.

Since for the algorithm the scene is just a set of pixels with feature values, in terms of computational costs all the scenes are the same. Therefore, the dependency of running time on the number of the scenes is linear.

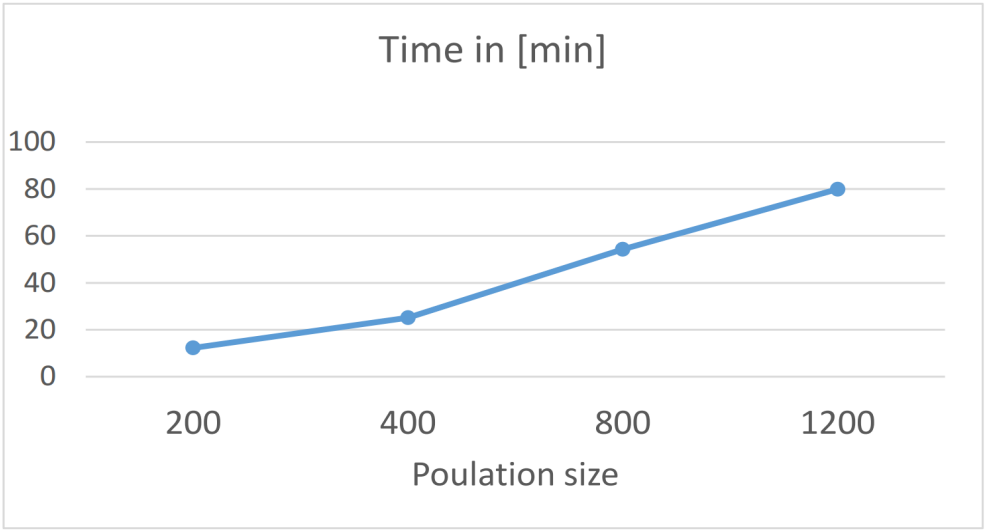


Figure 4.1: The dependency of the running time of the algorithm on the size of the population.

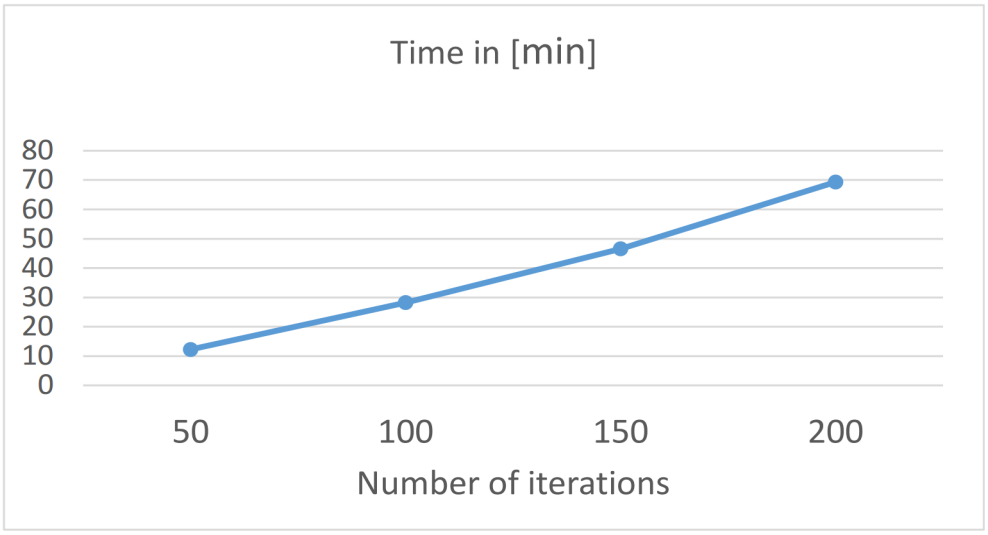


Figure 4.2: The dependency of the running time of the algorithm on the number of iterations.

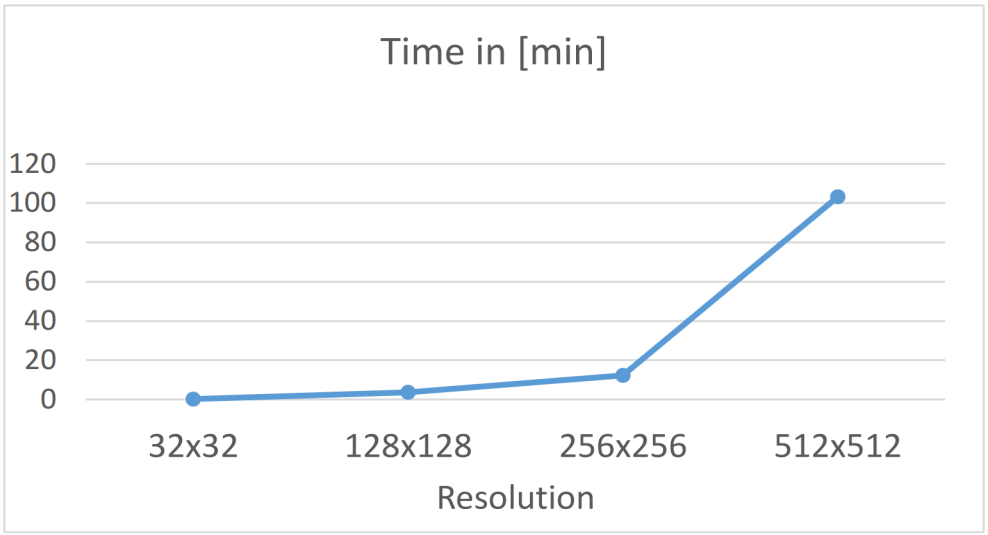


Figure 4.3: The dependency of the running time of the algorithm on the resolution of the scene.

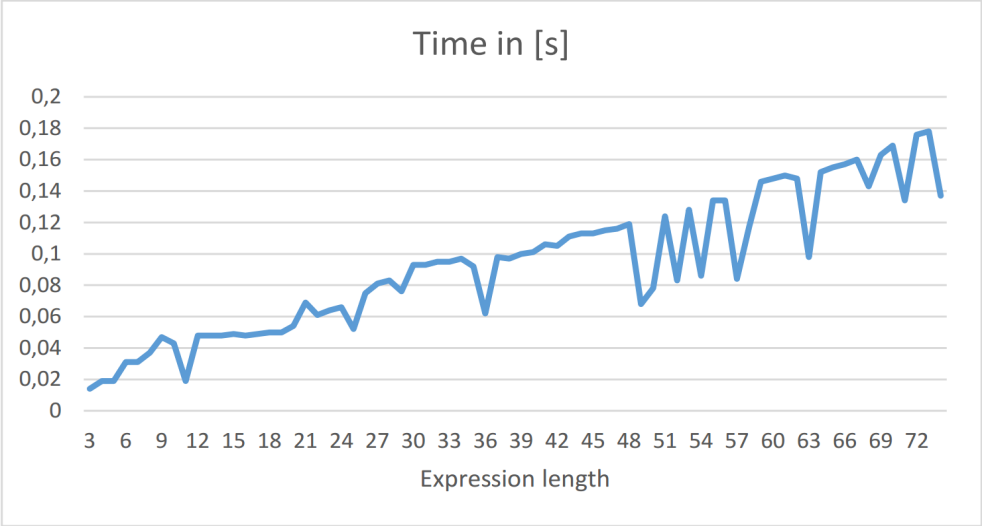


Figure 4.4: The dependency of the running time of expression's fitness assignment on the length of expression.

4.1.3 Performance Discussion

The algorithm spends most of the time in the fitness evaluation procedure. Time spent in one iteration for selection and expansion does not depend on the resolution and the number of scenes. It is only dependent on the size of population and the length of the expressions. But since we have the restriction on the maximum length of the expression and the population is not bigger than 2000, the time spent in such operations is only a tiny fraction of the second, as can be seen in Table 4.5. In contrast to that, fitness evaluation function is a bottleneck of the algorithm. Because of the recursive evaluation of the neighbour weights, the algorithm is not well suited for parallelization. Additionally, because of the random nature of the algorithm the expression can contain a lot of calculation involving features. Reading feature values in our implementation is performed through the reading of the texture values from the global memory and requests to the global memory are expensive in GPU.

4.2 Error Evaluation

The relMSE in figures 4.6 and 4.7 is calculated as the minimum of all relMSEs of expressions generated during the optimization. As it can be seen in Figure 4.6 the error is decreasing linearly with the increase of the iterations number. In contrast to that, in relation of population size and relMSE we can not observe such dependency. This can be explained by random nature of the algorithm, even tough with bigger population the algorithm has more variety, if optimization with smaller population size performs only one good mutation and obtains better expression, it will be propagated till the end.

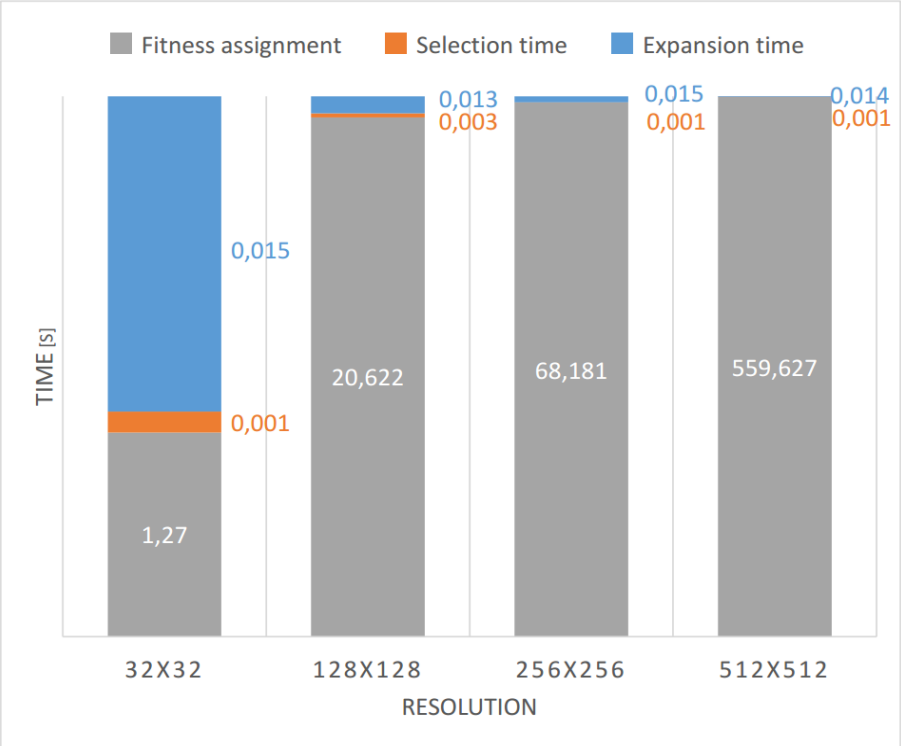


Figure 4.5: The time spend by one iteration of GP algorithm with different resolutions. The timing was obtained by running optimization on the teapots scene shown in Figure 1.1b with population of 4 islands and 300 individuals in each island.

Therefore, the result of optimization run with the smaller population can have lower relMSE than the optimization run with bigger population.

The dependency of relMSE on the number of scenes is complicated because we normalize the final fitness value by the number of scenes and the scenes at different resolution have different initial relMSE values.

As it can be seen from Table 4.2 the relMSE of the optimizaition's best expression depends on which expansion methods are used. Utilization of 90 % of all methods as crossover proposed by Koza [Koz92] does not work good in our case. The best results were achieved by the use of 45 % of crossovers, 45 % of mutations and 10 % of reproductions, to assure the propagation of good filtering expressions to the next generation.

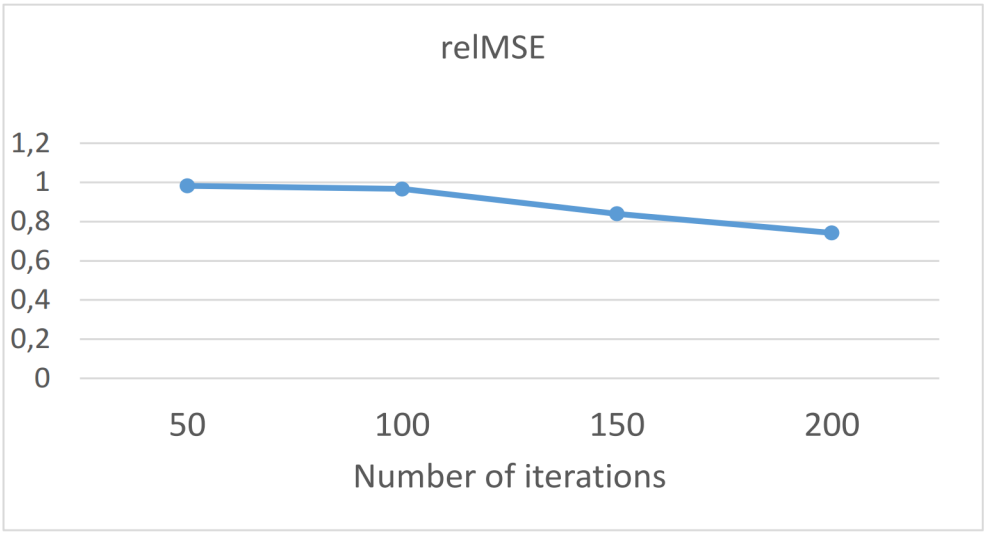


Figure 4.6: The dependency of the relMSE of the algorithm on the number of iterations.

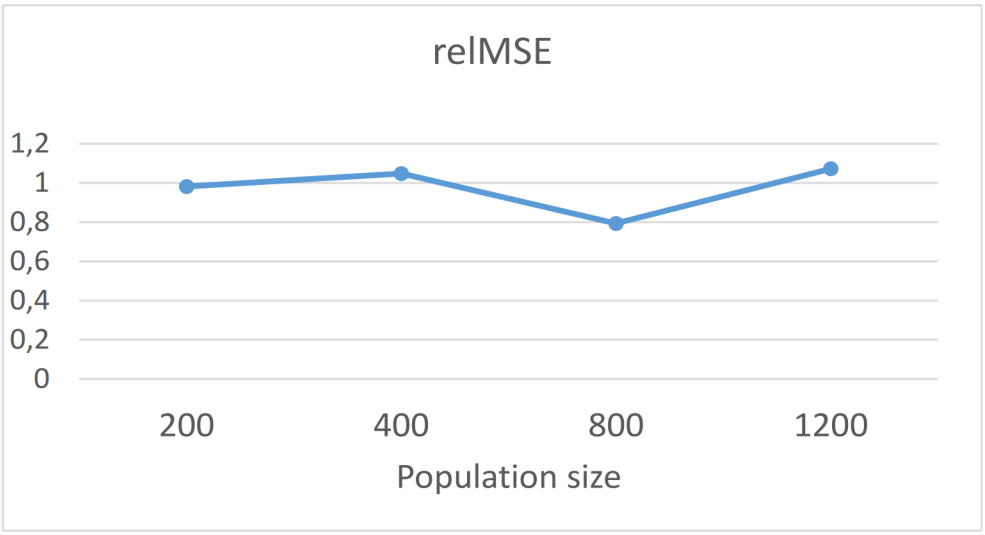


Figure 4.7: The dependency of the relMSE of the algorithm on the size of the population.

4.3 Empirical Investigation of Features

After performing numerous experiments with different parameters we can plot the frequencies of usage of different features during our optimization. These frequencies are shown in Figure 4.8. As can be seen in the figure the most frequently used feature is normal. The best set of features would be **normal, direct illumination, textures, depth, secondary texture gradient** and **world position**.

Different probabilities distribution	RelMSE
0.1 reproduction, 0.15 crossover, 0.75 mutation	0.98257
0.1 reproduction, 0.45 crossover, 0.45 mutation	0.377417
0.33 reproduction, 0.33 crossover, 0.34 mutation	0.70105
0.01 reproduction, 0.9 crossover, 0.09 mutation	1.16676

Table 4.2: Comparison of the relMSE obtained from the optimization run with different expansion methods probabilities.

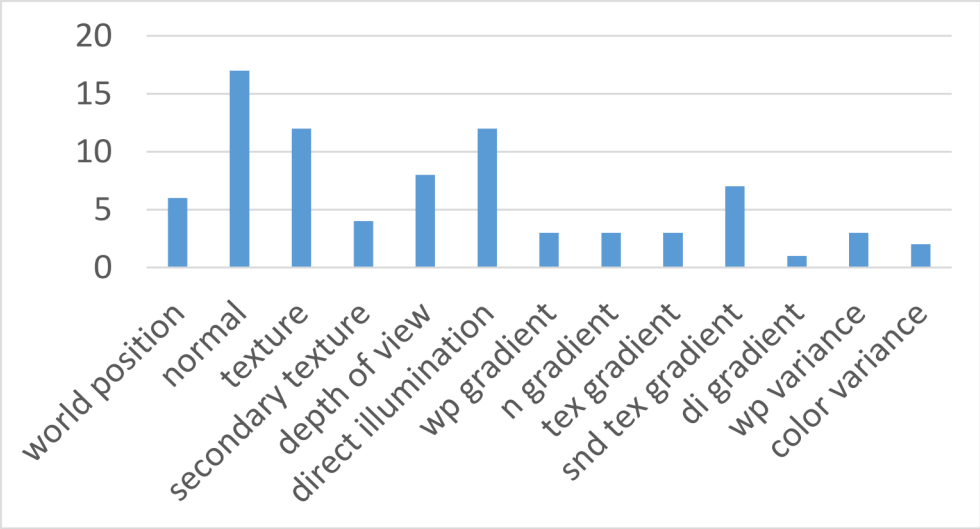


Figure 4.8: Frequencies of occurrences of different features after 20 experiments with different parameters and trained with different training scenes.

4.4 Qualitative Comparison of the Results

In this section we present novel high-dimensional filtering expressions, the best expressions obtained after numerous optimization runs of our algorithm. We compare the results of filtering with such expressions with the cross-bilateral (CB) filter with constant feature sigmas and with implementation of Learning Based Filtering (LBF) [KBS15]. To be able to compare the results with LBF the testing scenes were taken from the PBRT renderer. Expressions are presented as formulas of weight calculations for filtering equation (Equation 2.10).

First expression. First expression was obtained by running optimization with 3 training scenes from PBRT.

$$w_{ij}^1 = e^{-\left(||f_i^d - f_j^d||_{L2} + ||f_i^n - f_j^n||_{L2} + 2|f_i^t \cdot f_j^t| + ||c_i - c_j||_{L2} + ||f_i^t - f_j^t||_{max} + ||f_i^n - f_j^n||_{L1}\right)} \cdot \text{biweight}\left(-\left(\left(||f_i^{di} - f_j^{di}||_{L2}^2\right)^2 \cdot 2\right)\right) \quad (4.1)$$

where $||\cdot||_{L2}$ is Euclidean distance, $||\cdot||_{max}$ is Chebychev distance, $\cdot$ is dot product, biweight is biweight kernel (Equation 3.2), f_i^d is the depth feature value, f_i^n is the normal feature value, f_i^t is the texture feature value, f_i^{di} is the direct illumination feature value and c_i is the noisy color value of ith pixel. Evaluation of this expression can be seen in Figure 4.9. Cross-bilateral filter perform poorly on that scene. Relative MSE of our algorithm and LBF is comparable. The roughness of the material on the second floor of the scene and the edges are preserved better by our expression. The textures of paved stone of the floor are more blurred by our method than in LBF and some of the grained noise is distinguishable on the darkened parts of the scene.

Second expression Second expression was trained with all 6 training scenes:

$$w_{ij}^2 = e^{-atan(|c_i \cdot c_j|)} \cdot \text{second_exp} \quad (4.2)$$

where second_exp is

$$e^{-\left(\left(\left(||f_i^{di} - f_j^{di}||_{L1} ||f_i^{di} - f_j^{di}||_{max}\right)^{\text{pow}} \cos\left(e^{-\frac{||f_i^n - f_j^n||_{L2}^2}{0.13}}\right) + |f_i^t \cdot f_j^t|\right)\left(||p_i - p_j||_{L1} - atan\left(-||f_i^n - f_j^n||_{L1}\right)\right)\right)} \quad (4.3)$$

, pow is

$$2 + \text{biweight}\left(\sqrt{\frac{||f_i^{wpg} - f_j^{wpg}||_{L2} ||p_i - p_j||_{L2}}{0.01}}\right) + ||f_i^{wp} - f_j^{wp}||_{L1} \quad (4.4)$$

, $||\cdot||_{L1}$ is Manhattan distance, wp is world position and wpg is its gradient, p_i are coordinates of the ith pixel. Evaluation of this expression can be seen in Figure 4.10. Second expression outperform both LBF and CB in terms of error and LBF in the speed. But in terms of visual pleasantness the scene filtered with our expression is more blurred than scene filtered with LBF.

Third expression Third expression was trained with all 6 training scenes:

$$w_{ij}^3 = e^{-\frac{||f_i^{wp} - f_j^{wp}||_{max}}{0.05}} * \text{second_exp}_2 \quad (4.5)$$

where second_exp_2 is

$$e^{-\left(\frac{||p_i - p_j||_{L1} + 2 + e^{e^{sinc\left(||f_i^{wp} - f_j^{wp}||_{L2}^{0.002}\right)}}}{2^{asin\left(\text{biweight}\left(\left(\left(\frac{||f_i^n - f_j^n||_{L2}}{2}\right)^{\text{power}_1}\right)^{\text{power}_2}\right)\right)}}\right)} \quad (4.6)$$

, power_1 is

$$e^{\text{biweight}\left(||c_i - c_j||_{max} 2^{3 \cdot \text{sinc}\left(||f_i^{di} - f_j^{di}||_{L2}\right)}\right)} \tag{4.7}$$

, power_2 is

$$e^{-\frac{||f_i^t - f_j^t||^2_{max}}{0.01}} \cdot e^{\text{tricube}\left(||f_i^{di} - f_j^{di}||_{L2}\right) \cdot \text{mitchell}(var^{wp})} \tag{4.8}$$

, var^{wp} is variance of the world position feature. *sinc* (Equation 3.2.2), *tricube* (Equation 3.3) and *mitchell* (Equation 3.1) are unary operators from the codebook. Evaluation of this expression can be seen in Figure 4.11. San-Miguel scene with such point of view contains a lot of details and is complex. Cross-bilateral filter leaves a lot of noise on that scene. Even tough relMSE error in our method is less than LBF, it overblurs the small details, as vases and plates, and the shadow of the bulk near the roof.

All three expressions perform relatively good. They outperform the cross-bilateral filter in all of the cases in terms of quality and have lower relMSE than LBF in two scenes. In terms of timing our expressions outperform significantly LBF on all of the testing scenes and have comparable time with CB. First expression is trained only on the PBRT scenes, it preserves edges better than other two expressions but does not remove all the grained noise from the scenes. In contrast to that, remaining two expressions are trained on all of the training scenes, obtained both from PBRT and Optix, and remove grained noise from the image. The disadvantage of these two expressions is that they in some cases overblur without preservation of all of the edges.

4.5 Discussion

Genetic Programming algorithm is capable of deriving novel filtering expressions. With the assistance of the features it can generate expressions that can outperform the cross-bilateral filter in terms of quality and LBF in terms of time. Because the main algorithm's evaluation criterion on expression's quality is relMSE, what is not the same as assessment by the human eye, our results could not outperform state of the art in terms of subjective visual quality. The biggest hurdle for the use of GP algorithm in our case was time spent on fitness evaluation of the expressions. It poses restriction on the resolution and the amount of the training scenes that can be used for the optimization, because in such cases the running time of the algorithm becomes too long for multiple experiments.

(a) Noisy input, relMSE = 2.5267

(b) CB, relMSE = 1.773, time = 0.407s

(c) Ours, relMSE = 0.8567, time = 0.483s

(d) LBF, relMSE = 0.762053, time = 5.79s

(e) Ground truth, time = 24 hours

Figure 4.9: Evaluation of first expression. The noisy input was rendered using 4 samples per pixel at the 512x512 resolution. Sponza model is the courtesy of Marko Dabrovic and Mihovil Odak.

(a) Noisy input, relMSE = 3.41

(b) CB, relMSE = 0.6482, time = 0.846s

(c) Ours, relMSE = 0.3017, time = 0.883s

(d) LBF, relMSE = 0.50219, time = 9.468s

(e) Ground truth, time = 48 hours

Figure 4.10: Evaluation of second expression. The noisy input was rendered using 4 samples per pixel at the 800x550 resolution. The scene - San Miguel is the courtesy of Guillermo M. Leal Llaguno.

(a) Noisy input, relMSE = 6.847

(b) CB, relMSE = 1.1282, time = 0.8s

(c) Ours, relMSE = 0.3875, time = 0.852s

(d) LBF, relMSE = 0.4016, time = 9.513s

(e) Ground truth, time = 48 hours

Figure 4.11: Evaluation of third expression. The noisy input was rendered using 4 samples per pixel at the 800x550 resolution. The scene - San Miguel is the courtesy of Guillermo M. Leal Llaguno.

CHAPTER 5

Limitations and Future Work

Search Space Limitations. Expressiveness is an ability of possible equations in search space to represent the optimal solution. We use strongly typed genetic programming (STGP) [Mon95] in our method. STGP is a genetic programming algorithm where each terminal value is assigned to a certain type and functional operators can take only certain types of terminal values as arguments. Such technique suits our method well, because we have scalar variables and constants with features, which are represented as vectors, and we would like to use different operators on each of them. STGP limits the search space of the possible solutions, by introducing the constraints on the alternation of nodes in the AST. Further limitations of the search space are the initial population and the expressions in the codebook. It is only possible to obtain a high-dimensional filter, that uses features, coordinates and colors as the means for assignment of weights. Therefore, our GP can not derive something like frequency domain filters with utilization of wavelet and Fourier transform. The more variety the codebook has, the bigger the search space will be.

Expression's Bloat. The algorithm is prone to the problem of expression's bloat, the growth of expression without significant improvement of fitness value. One possible solution to that would be to use Pareto frontier for fitness value assignment in every iteration. The problem of Pareto frontier is that, if we make a graph depending on expression's length and fitness value some of the points on that graph would not be formally comparable, take for example an expression of length 60 with fitness value of 0.5 and an expression of length 30 with fitness value of 0.87. Therefore, it would require manual selection of the fittest after every iteration, what is not admissible in our case. The problem of big expressions is the lack of demonstrativeness. Because of the random nature of the algorithm big expression is hard to interpret, as they represent only big sequence of operations with no logic behind it. Additionally, the bigger the expression is, the bigger space should be investigated for optimization. It then takes more iterations for tangible improvement of fitness value.

Generality of the Approach Our GP optimization does not include effects such as motion blur, depth of the field and participating media. Extracted features from such scenes contain high level of noise and therefore, they can not be used as guiding means for the filtering. Possible solution to that would be either prefiltering, for example with NLM filter [RKZ12] or utilization of modified distance function [LWC12] by dividing the distance by the sum of sample variances of variables. The problem with prefiltering the features during rendering is its additional computational overhead, what would significantly slow down the filtering process.

Implementation Limitations. We had to restrict the maximum length of the expression to 75, because GPU can not evaluate longer expressions in one pass. Despite constraining the size of the search space this restriction also solves a problem of uncontrollable bloat of the expressions. As a tradeoff between efficiency and computational time the maximum number of iterations is set to 200. Because of the very long computational time of algorithm for big resolutions (around four to five days for 512x512 resolution and population of 400 individuals) the further improvement of fitness value by increasing the number of iterations was not investigated.

Computation of the Filter Parameters. Another limitation of our algorithm is the absence of ability to change the filter width or feature weights. We introduced the sample variance in form of scalars to the algorithm to give GP optimization an opportunity to use them as parameters. For example GP algorithm can exchange color variance for constant sigma in cross-bilateral filter. The work of Kalantari *et al.* [KBS15] uses neural network to look for relation between statistical data extracted from features and filter parameters. Taking into account the generality of the genetic programming it is worth trying to look for explicit expressions to couple this statistical data with feature weights. In addition to our main genetic programming algorithm we performed an experiment for optimizing the expressions for feature weights calculation in cross-bilateral filter. The statistical data of features that were used were mean and standard deviation, gradient, mean deviation and sampling rate. The unary and the binary operators were the same as in our main GP implementation. The only functional operator was length of the feature vector. As there are no known expressions for such relation no codebook could be used in the optimization. The population was initialized with the sum of the scalar variables and the length of the gradient of feature. The final result of the experiment was not successful and after numerous iterations fitness value stuck in the local minimum. One of the possible shortcomings could be the banal initialization of the population. Since there is no good starting point for the search the ramped half-and-half would be more preferable in this case. The possible application for that method is improvement of our basic GP algorithm. We can introduce the obtained expressions for feature weights as variables, which can work as adjusting parameters for the generated expressions in every pixel for different scenes. For example, if the variance at the certain pixel for the normal is big and the normal weight is proportional to the variance, the expression will lower the significance of that feature during neighbour weight calculation.

Combination of Filters. Another possible future work is combination of filters trained for the specific effects, by adding them to the filter bank. During the filtering phase we can then apply the specific filtering expression on the neighbourhood of the pixel, where certain effect was identified. It may require additional information such as per sample depth or motion information, but it should not be much of the computational overhead.

CHAPTER 6

Conclusion

We presented a framework for searching novel filtering expressions with the utilization of GP algorithm. We found out that GP algorithm is applicable for the filtering and has only one big limitation - long fitness evaluation time. The end result of our framework are the novel high-dimensional filtering expressions. The best expressions obtained from the experiments are better in terms of quality than cross-bilateral filter with constant sigmas for features. Filtering implementations of such expressions on the GPU in Optix are significantly faster than state of the art filtering algorithm presented by Kalantari *et al.* [KBS15]. Despite of not trying to train expressions on effects such as depth of field and motion blur due to the noisiness of the features, it is possible to use such scenes for training if we prefilter features by NLM filter [RKZ12] or use modified distance function [LWC12]. Therefore, our framework can be generalized for many effects in Monte Carlo rendering. The search space of the algorithm is wide and restrained only by our grammar. Potentially, with the big number of good training scenes and excessive amount of iterations other very good filtering expressions could be obtained.

Additionally, after the run of the numerous experiments we investigated the occurrences of the features in the final results of the optimizations. According to our results the most frequently used features are normal, texture and direct illumination. The application of the GP for filtering of Monte Carlo noise is a novel approach with the promising results.

Bibliography

[AB06] Remi Arnaud and Mark C. Barnes. *Collada: Sailing the Gulf of 3D Digital Content Creation*. AK Peters Ltd, 2006.

[ABD10] Andrew Adams, Jongmin Baek, and Myers Abraham Davis. Fast high-dimensional filtering using the permutohedral lattice. In *Computer Graphics Forum (EG 2010 Proceedings)*, page 2010, 2010.

[ABI98] Ricardo Aler, Daniel Borrajo, and Pedro Isasi. Genetic programming and deductive-inductive learning: A multi-strategy approach. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 10–18, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[AW95] Volker Aurich and Jörg Weule. Non-linear gaussian filters performing edge preserving diffusion. In *Mustererkennung 1995, 17. DAGM-Symposium*, pages 538–545, London, UK, UK, 1995. Springer-Verlag.

[Bak89] J. E. Baker. *An Analysis of the Effects of Selection in Genetic Algorithms*. PhD thesis, Nashville, TN, USA, 1989. UMI order no: GAX89-19677.

[BCM05] A. Buades, B. Coll, and J. M. Morel. A review of image denoising algorithms, with a new one. *Simul*, 4:490–530, 2005.

[BLPW14] Adam Brady, Jason Lawrence, Pieter Peers, and Westley Weimer. genbrdf: Discovering new analytic brdfs with genetic programming. *ACM Trans. Graph.*, 33(4):114:1–114:11, July 2014.

[BT96] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evol. Comput.*, 4(4):361–394, December 1996.

[CPC84] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18(3):137–145, January 1984.

[CPD07] Jiawen Chen, Sylvain Paris, and Frédo Durand. Real-time edge-aware image processing with the bilateral grid. *ACM Trans. Graph.*, 26(3), July 2007.

[DD02] Frédo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Trans. Graph.*, 21(3):257–266, July 2002.

[DSHL10] Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik P. A. Lensch. Edge-avoiding À-trous wavelet transform for fast global illumination filtering. In *Proceedings of the Conference on High Performance Graphics*, HPG '10, pages 67–75, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[EHDR11] Kevin Egan, Florian Hecht, Frédo Durand, and Ravi Ramamoorthi. Frequency analysis and sheared filtering for shadow light fields of complex occluders. *ACM Trans. Graph.*, 30(2):9:1–9:13, April 2011.

[Epa69] V. A. Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theory of Probability & Its Applications*, 14(1):153–158, 1969.

[ETH+09] Kevin Egan, Yu-Ting Tseng, Nicolas Holzschuch, Frédo Durand, and Ravi Ramamoorthi. Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Trans. Graph.*, 28(3):93:1–93:13, July 2009.

[FM95] G. J. Ferrer and W. N. Martin. Using genetic programming to evolve board evaluation functions. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 2, pages 747–752 vol.2, Nov 1995.

[GO11] Eduardo S. L. Gastal and Manuel M. Oliveira. Domain transform for edge-aware image and video processing. *ACM Trans. Graph.*, 30(4):69:1–69:12, July 2011.

[GO12] Eduardo S. L. Gastal and Manuel M. Oliveira. Adaptive manifolds for real-time high-dimensional filtering. *ACM Trans. Graph.*, 31(4):33:1–33:13, July 2012.

[Gro85] Paul Bryant Grosso. *Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model*. PhD thesis, Ann Arbor, MI, USA, 1985. AAI8520908.

[Has70] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[hChY97] Shu heng Chen and Chia hsuan Yeh. Using genetic programming to model volatility in financial time series: The cases of nikkei 225 and s&p 500. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 288–306. Morgan Kaufmann, 1997.

[HJW+08] Toshiya Hachisuka, Wojciech Jarosz, Richard Peter Weistroffer, Kevin Dale, Greg Humphreys, Matthias Zwicker, and Henrik Wann Jensen. Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph.*, 27(3):33:1–33:10, August 2008.

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

[HST10] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. In *Proceedings of the 11th European Conference on Computer Vision: Part I*, ECCV'10, pages 1–14, Berlin, Heidelberg, 2010. Springer-Verlag.

[Ká14] Peter Kán. *High-Quality Real-Time Global Illumination in Augmented Reality*. PhD thesis, Kán, 2014.

[Kaj86] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[KBS15] Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. A machine learning approach for filtering monte carlo noise. *ACM Trans. Graph.*, 34(4):122:1–122:12, July 2015.

[KMA+15] Markus Kettunen, Marco Manzi, Miika Aittala, Jaakko Lehtinen, Frédo Durand, and Matthias Zwicker. Gradient-domain path tracing. *ACM Trans. Graph.*, 34(4):123:1–123:13, July 2015.

[Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[KS13] Nima Khademi Kalantari and Pradeep Sen. Removing the noise in Monte Carlo rendering with general image denoising algorithms. *Computer Graphics Forum (Proceedings of Eurographics 2013)*, 32(2), 2013.

[LAC+11] Jaakko Lehtinen, Timo Aila, Jiawen Chen, Samuli Laine, and Frédo Durand. Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph.*, 30(4):55:1–55:12, July 2011.

[LALD12] Jaakko Lehtinen, Timo Aila, Samuli Laine, and Frédo Durand. Reconstructing the indirect light field for global illumination. *ACM Trans. Graph.*, 31(4):51:1–51:10, July 2012.

[LW93] Eric P Lafortune and Yves D Willems. Bi-directional path tracing. In *Proceedings of CompuGraphics*, volume 93, pages 145–153, 1993.

[LWC12] Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. Sure-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graph.*, 31(6):194:1–194:9, November 2012.

[MCY14] Bochang Moon, Nathan Carr, and Sung-Eui Yoon. Adaptive rendering based on weighted local regression. *ACM Trans. Graph.*, 33(5):170:1–170:14, September 2014.

[MKSS14] Victor May, Yosi Keller, Nir Sharon, and Yoel Shkolnisky. An algorithm for improving non-local means operators via low-rank approximation. *CoRR*, 4, 2014.

[MN88] Don P. Mitchell and Arun N. Netravali. Reconstruction filters in computer-graphics. *SIGGRAPH Comput. Graph.*, 22(4):221–228, June 1988.

[Mon95] David J. Montana. Strongly typed genetic programming. *Evol. Comput.*, 3(2):199–230, June 1995.

[MVZ16] M. Manzi, D. Vicini, and M. Zwicker. Regularizing image reconstruction for gradient-domain rendering with feature patches. *Computer Graphics Forum*, 35(2):263–273, 2016.

[ODR09] Ryan S. Overbeck, Craig Donner, and Ravi Ramamoorthi. Adaptive wavelet rendering. *ACM Trans. Graph.*, 28(5):140:1–140:12, December 2009.

[PBD+10] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. Optix: a general purpose ray tracing engine. *ACM Transactions on Graphics (TOG)*, 29(4):66, 2010.

[PD09] Sylvain Paris and Frédo Durand. A fast approximation of the bilateral filter using a signal processing approach. *Int. J. Comput. Vision*, 81(1):24–52, January 2009.

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010.

[PLM08] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming.* Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[PSA+04] Georg Petschnigg, Richard Szeliski, Maneesh Agrawala, Michael Cohen, Hugues Hoppe, and Kentaro Toyama. Digital photography with flash and no-flash image pairs. *ACM Trans. Graph.*, 23(3):664–672, August 2004.

[RKZ11] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graph.*, 30(6):159:1–159:12, December 2011.

[RKZ12] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. Adaptive rendering with non-local means filtering. *ACM Trans. Graph.*, 31(6):195:1–195:11, November 2012.

[RMZ13] Fabrice Rousselle, Marco Manzi, and Matthias Zwicker. Robust denoising using feature and color information. In *Computer Graphics Forum*, volume 32, pages 121–130. Wiley Online Library, 2013.

[SAMWL11] Pitchaya Sitthi-Amorn, Nicholas Modly, Westley Weimer, and Jason Lawrence. Genetic programming for shader simplification. *ACM Trans. Graph.*, 30(6):152:1–152:12, December 2011.

[SB97] Stephen M. Smith and J. Michael Brady. Susan—a new approach to low level image processing. *Int. J. Comput. Vision*, 23(1):45–78, May 1997.

[SD12] Pradeep Sen and Soheil Darabi. On filtering the noise from the random parameters in monte carlo rendering. *ACM Trans. Graph.*, 31(3):18:1–18:15, May 2012.

[Sie76] David Siegmund. Importance sampling in the monte carlo study of sequential tests. *The Annals of Statistics*, pages 673–684, 1976.

[SSD+09] Cyril Soler, Kartic Subr, Frédo Durand, Nicolas Holzschuch, and François Sillion. Fourier depth of field. *ACM Trans. Graph.*, 28(2):18:1–18:12, May 2009.

[SZR+15] Pradeep Sen, Matthias Zwicker, Fabrice Rousselle, Sung-Eui Yoon, and Nima Khademi Kalantari. Denoising your monte carlo renders: Recent advances in image-space adaptive sampling and reconstruction. In *ACM SIGGRAPH 2015 Courses*, SIGGRAPH '15, pages 11:1–11:255, New York, NY, USA, 2015. ACM.

[TM98] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, pages 839–, Washington, DC, USA, 1998. IEEE Computer Society.

[Whi79] Turner Whitted. An improved illumination model for shaded display. In *ACM SIGGRAPH Computer Graphics*, volume 13, page 14. ACM, 1979.